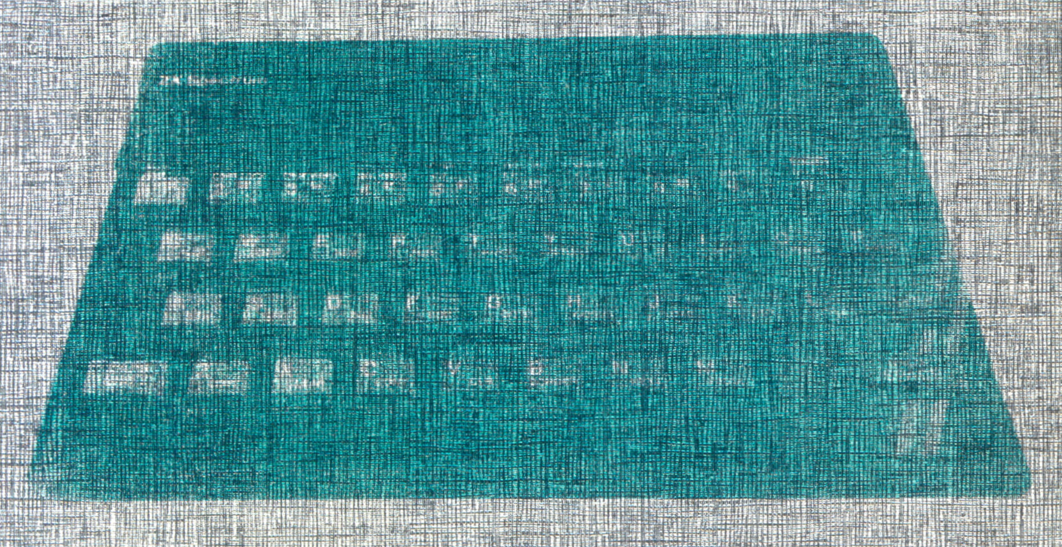
ZX Spectrum
(TS 2068)
Técnicas de *Procesamiento de la Información*

C. A. Street





ZX SPECTRUM (TS 2068)
TECNICAS DE *PROCESAMIENTO DE LA INFORMACION*

CONSULTORES EDITORIALES
AREA DE INFORMATICA Y COMPUTACION

Antonio Vaquero Sánchez

Catedrático de Informática
Facultad de Ciencias Físicas
Universidad Complutense de Madrid
ESPAÑA

Isaac Schnadower

Departamento de Electrónica
Universidad Autónoma Metropolitana
Gerente General de Servicios
Educativos Computacionales
MEXICO

Alfonso Pérez Gama

Ingeniero Electrónico
Universidad Nacional de Colombia
COLOMBIA

José Portillo

Universidad de Lima
PERU

ZX Spectrum (TS 2068)
Técnicas de **Procesamiento de la Información**

C.A. STREET

Traducción

Luis Joyanes Aguilar

Capitán de Artillería
Licenciado en Ciencias Físicas
Grupo Electrónica
Academia de Artillería de Madrid

REVISION TECNICA

Antonio Vaquero Sánchez

Catedrático de Informática
Facultad de Ciencias Físicas
Universidad Complutense de Madrid

McGraw-Hill

MADRID • BOGOTÁ • BUENOS AIRES • GUATEMALA • LISBOA • MÉXICO
NUEVA YORK • PANAMÁ • SAN JUAN • SANTIAGO • SAO PAULO
AUCKLAND • HAMBURGO • JOHANNESBURGO • LONDRES • MONTREAL
NUEVA DELHI • PARÍS • SAN FRANCISCO • SINGAPUR
ST. LOUIS • SIDNEY • TOKIO • TORONTO

ZX SPECTRUM (TS 2068) TECNICAS DE PROCESAMIENTO DE LA INFORMACION

Atlacomulco, 499-501, Naucalpan de Juárez, Edo. de México
Miembro de la Cámara Nacional de la Industria Editorial, Reg. Num. 465

ISBN: 968-451-725-4

Traducido de la primera edición en inglés de

INFORMATION HANDLING FOR THE ZX SPECTRUM

ISBN: 0-07-084-707-X

Edición exclusiva para Ediciones La Colina, S. A. (España)

ISBN: 84-7615-014-8
Depósito legal: M. 11.080-1985

Gráficas EMA. Miguel Yuste, 27. 28037 Madrid

PRINTED IN SPAIN-IMPRESO EN ESPAÑA

INDICE GENERAL

Capítulo 5
COMPARACION Y CLASIFICACION

Capítulo 6
MANTENIMIENTO DE SUS FICHEROS

APENDICE

PROLOGO

Este libro está concebido para quienes deseen hacer uso de sus computadoras a pequeña escala, pero de una forma útil. Probablemente, la aplicación más común para las grandes computadoras es la de almacenamiento, a gran escala, de información de todas clases (desde el nacimiento hasta nuestros detalles personales, estados financieros, actividades profesionales y otros hechos son objeto de registro y utilización en sistemas de computadoras grandes). En términos generales, esta actividad mejora la calidad de vida para el individuo y disminuye y alivia la carga de trabajo (sobre todo en lo que respecta a sus aspectos más rutinarios) en grandes organismos privados y públicos. El procesamiento de datos computarizado, adecuadamente controlado y supervisado, "engrasa las ruedas" de la vida moderna con menos implicaciones funestas que las que son inherentes a muchas otras invenciones del siglo XX.

Aunque la computadora personal sea más lenta y tenga menos capacidad de almacenamiento que sus parientes más grandes, siguen pudiendo proporcionar la misma exactitud y comodidad para los usuarios domésticos. He tratado de proporcionar dos cosas útiles en este libro. En primer lugar, los programas son construcciones de trabajo que se pueden introducir desde el teclado o desde una cinta y se pueden utilizar de forma inmediata. En la última parte del libro, se han concebido de forma que partes de un programa se pueden utilizar en otro; de este modo, una vez que se hayan comprendido los diversos procesos, el lector debe ser capaz de utilizar módulos procedentes de dos o más programas individuales para poder construir sistemas adaptados individualmente a sus propias necesidades especiales. Con el mismo fin, me he esforzado en conseguir que cada módulo sea lo más flexible posible, de modo que el cambio de un programa concebido para el almacenamiento de nombres y de direcciones a otro para, por ejemplo, un catálogo de sellos debe ser una operación rápida y sencilla.

En segundo lugar, he intentado utilizar un método "estructurado" para planificar los programas. La esencia de este método es que el problema se estudia a fondo antes de que se intente cualquier codificación (el acto de escribir las propias sentencias de BASIC). Una vez analizado el problema, una descripción de sus métodos se escribe en "pseudocódigo" que conduce luego a la versión de BASIC final. El resultado debe ser unos programas que son más fáciles de corregir y de asimilar. Si el programa se lee conjuntamente con la descripción en pseudocódigo (los principios de este procedimiento se introducen en el primer Capítulo) sus elementos

de trabajo deben ser claros. Espero que mis lectores estarán de acuerdo. Este libro no es, sin embargo, un texto más sobre programación estructurada; después de la breve introducción en el Capítulo 1, se utiliza a través de todo el libro en el curso de la escritura de programas que funcionan adecuadamente y que son de utilidad. Tengo la esperanza de que al escribir de esta manera habré evitado la trampa de programas artificiales con miras a una práctica de programación teóricamente correcta. He adquirido la mayor parte de mis conocimientos de programación a través de los intentos de hacer algo útil con una máquina, en muchos casos siguiendo y adaptando programas creados por expertos. Es mi deseo que los lectores adquieran unos beneficios semejantes a partir del trabajo con los programas incluidos en este libro. Para conseguir una utilidad máxima, se precisa disponer de una ZX Spectrum de 48K, pues es mucho lo que se puede conseguir adaptando programas desde un dialecto de BASIC a otro, lo que constituye una tarea que lleva bastante tiempo pero que resulta ser una práctica muy instructiva.

En el Capítulo 2, he tratado de abordar lo que, para mí, es la parte más importante y difícil del lenguaje BASIC. La construcción de IF...THEN es aparentemente sencilla, pero tiene trampas ocultas, y espero que una lectura detenida ayudará a los lectores a reconocer y evitar las trampas que les acecha. En lo sucesivo, los programas se concentran en el objetivo principal: almacenamiento, recuperación, clasificación y corrección de la información en un ZX Spectrum de 48K estándar, con el empleo de cinta de casete para almacenamiento permanente. Muchos de los métodos están relacionados con los utilizados en el programa PROFILE (perfil), que es un programa de manipulación de ficheros publicado también por McGraw-Hill pero se precisaría un libro de mucha mayor extensión para exponer todo lo referente a este programa, que se concibió como una herramienta de trabajo con el empleo de algunas técnicas de programación que van más allá del alcance de este libro.

Por supuesto, no es mi intención que los programas sean perfectos, de hecho, me he abstenido intencionadamente de utilizar algunas de las características que pueden hacer más atractivo cualquier programa. El color y el sonido, por ejemplo, se utilizan muy pocas veces, pero ello fue una decisión premeditada con el fin de permitir que la atención se concentre en las partes esenciales de la manipulación de los datos. Un programa bien construido siempre puede tener elementos de atracción añadidos, como un colofón final, pero si el diseño comienza con estos elementos, puede ser que no realice con eficacia sus funciones básicas.

Finalmente, quiero expresar mi agradecimiento a todas aquellas personas cuya ayuda paciente ha contribuido en gran medida a la

elaboración de esta obra. En particular, debo mencionar a Grahan Bishop por su gran estímulo y, por supuesto, a mi esposa Irene, sin cuyo apoyo no hubiera sido posible la elaboración del programa PROFILE ni la creación de este libro.

1 PROGRAMACION Y PLANIFICACION

La programación de una computadora es una actividad que produce satisfacción. Plantea un reto directo y coactivo a nuestras aptitudes de organización, perseverancia y raciocinio. La computadora personal es, por supuesto, una herramienta polifacética (con el software correcto, puede entretener, informar y actuar como un archivador electrónico), pero también puede comprometer a nuestros cinco sentidos en una tentativa para conseguir que realice nuestro deseo particular. Este libro está concebido para servir de ayuda a quienes hayan asimilado los elementos de programación en BASIC y quieran avanzar a un nivel en el que sus proyectos se puedan terminar con mayor rapidez, trabajando de forma más eficaz y facilitando las modificaciones, si cambiasen las circunstancias.

Todo ha de comenzar con el dominio del vocabulario del lenguaje de computadora y, como un reto, su sintaxis o reglas gramaticales. Una de las excelencias del Sinclair BASIC es su separación de los procesos de comprobación de la sintaxis (realizada tan pronto como se pulse la tecla ENTER) y la ejecución (después de RUN). El mensaje "syntax error" que se las ingenia para aparecer en el momento más inoportuno en otras máquinas, durante la ejecución de un programa, se sustituye por un amistoso, pero insistente, signo de interrogación que hace su aparición inmediatamente después de que se termine una línea de BASIC. Espero que a mis lectores no les importará si divago más en favor de la versión de BASIC utilizada en este libro (algunas de las puntualizaciones que hago han sido esenciales para la elaboración de los programas aquí encontrados y para el proyecto PROFILE asociado). Espero que los programas serán de utilidad en, como mínimo, dos formas: son aplicables para mostrar algunas de la técnicas para un eficaz almacenamiento y extracción de datos, pero cuando se prueban, y los programas de trabajo lo han de hacer, con el mínimo de adaptación, satisfacen muchas necesidades, sobre todo del tipo de "Archivadores".

Una segunda característica singular del Spectrum BASIC es el sistema de entradas de "palabras clave", que es idóneo para los jóvenes y para quienes escribiendo a máquina, como es mi caso, no han pasado nunca más allá de la etapa de los "dos dedos". Quizás

sea más importante el hecho de que el teclado se haga un diccionario del lenguaje, aunque sin definiciones, por lo que casi se elimina el problema enojoso de tener que examinar el manual para una función que sabe que está en cualquier lugar (si solamente...). Un subproducto de la entrada de palabras clave es que las palabras están separadas, de forma adecuada, con espacios como en cualquier otro elemento de trabajo impreso. Los lectores de revistas de computadoras sabrán cómo se confunden las versiones condensadas de programas a veces encontradas y así podría ser:

```
10FORK = STORE:PRINTED+K(S):NEXTK
```

en lugar de

```
10 FORK = sTOre:PRINTed + k(s):NEXTk
```

en lo que no solamente ayuda la separación entre palabras, sino también el convenio de que:

MAYUSCULAS para las palabras de BASIC
minúsculas para todo lo demás

que utilizaré a través de este libro, con una sola excepción. La única ocasión para desviarse de esta regla es cuando se introduce texto entre comillas en cadenas y sentencias PRINT, en donde suelen ser deseables las mayúsculas. La segunda regla utilizada al obtener versiones de más programas para este libro fue adoptada también con miras a la legibilidad. Se han insertado espacios en donde sean necesarios para facilitar la lectura y la copia en la medida de lo posible. En particular, las sentencias múltiples después de un número de línea se han separado de manera que se contenga cada sentencia en una sola línea de pantalla y para evitar el mal efecto de tener palabras divididas entre una línea y la siguiente. No hay otro motivo que no sea la legibilidad; en realidad, los programas se ejecutan un poco más lentamente en un formato expandido, por lo que si les copia puede desear teclearles en un flujo continuo. En los programas posteriores, ello sería ciertamente deseable pues los espacios suplementarios consumen memoria interna que se utiliza también para el almacenamiento de datos. Todos los programas se han ejecutado y probado en un Spectrum y fueron objeto de salida impresa por los editores en una impresora de matrices de puntos directamente a partir de las versiones de trabajo. Si el lector quiere ahorrarse la molestia de teclear los programas por sí mismo, puede conseguir una casete de todos los trabajos más importantes contenidos en este libro.

Hay otras dos características de Sinclair BASIC que le hacen, a mi juicio, la mejor versión actual del lenguaje para principiantes que se ha utilizado hasta ahora, lo cual no quiere decir que no pueda mejorarse todavía más. Ni una ni otra característica se pone

de manifiesto de forma inmediata, pero con una utilización continuada ambas han probado ser de gran utilidad. Todos los dialectos del lenguaje almacenan los valores de las variables en memoria, mientras se está ejecutando un programa, pero la versión de Sinclair tiene la singularidad de que cuando se modifica un programa no se pierden los valores, sino que simplemente, como el programa, se desplazan un poco de forma interna. Por supuesto, si luego teclea RUN se destruirán para siempre, pero si, en cambio, se emplea GOTO, se retendrán y reutilizarán los valores antiguos. Para muchas aplicaciones ello importa poco, pero cuando se trabaja con una masa de información que pueda haber llevado mucho tiempo introducir, esta característica tiene una gran importancia. Se extiende al almacenamiento de programas en cinta de casete (SAVE da como salida las líneas de programas y las variables para la cinta) y por ello, siempre que se tenga cuidado, nunca se perderán datos importantes que se necesiten. Ello es especialemte adecuado cuando se están desarrollando programas (los datos "introducidos" para fines de prueba pueden conservarse y utilizarse una y otra vez casi tan fácilmente como si se almacenara en un sistema de disco).

La última característica de Sinclair BASIC, que debo destacar, es la forma en que se ha mantenido lo más libre posible de las reglas que, en esencia, dice "No hacer...". En tanto que una sentencia haya aprobado la prueba del comprobador de la sintaxis, este dialecto hará todo lo posible para producir lo que piensa que el usuario desea, sin que lleve demasiado la concesión. A veces, ello significará llegar a longitudes estrambóticas. Pruebe el siguiente programa para ver lo que quiero decir:

```
10 LEFT f$ = "VAL$ f$": PRINT VAL$ f$
```

La máquina llena su memoria interna completa mientras se intenta resolver este problema imposible (Se trata de una definición circular "negro es negro, ¿qué es negro?"). Como una cuestión aparte, aunque muchos revisores hayan encontrado poco uso para la función VAL$, considero personalmente que es uno de los artilugios más elegantes del pequeño intérprete e indicará cómo se le puede utilizar en un capítulo posterior.

1.1 Velocidad

Está fuera de toda discusión el hecho de que el intérprete de Sinclair BASIC (el programa que traduce un conjunto de sentencias de BASIC en código de máquina y luego realiza las instrucciones resultantes), es lento en comparación con muchos otros. En mi opinión, ello no es perturbador y es, en parte, una consecuencia nece-

saria de los atributos positivos antes descritos. Otros intérpretes, por ejemplo, realizan el seguimiento de sus variables empleando conjuntos sofisticados de punteros que no pueden actualizarse cada vez que un cambio en el programa obligue a las variables a desplazarse internamente. Los sistemas de punteros son muy rápidos, pero relativamente inflexibles para ciertos fines. No obstante, la velocidad es muy útil, en algunas ocasiones, en un programa y trataré de poner de manifiesto cómo puede obtenerse la prestación necesaria cuando sea importante. Sin embargo, unas pocas reglas generales podrían ser de interés ahora.

1. Mantener en un mínimo el número de GOSUBs y de GOTOs. Cada vez que el intérprete encuentra una de ellas ha de buscar el objetivo, lo que lleva tiempo. Esta regla es importante, sobre todo, si la sentencia GO está en la parte media de un proceso repetitivo, en donde un retardo de unos pocos milisegundos cada vez que se encuentra la sentencia ha de multiplicarse por el número de veces que se repite el proceso. Esta es una regla excelente por otro motivo. Los programas con muchos saltos pronto se hacen difíciles de comprender debido a su aparente complejidad. En el Capítulo 2 le mostraré cómo, en muchos casos, las sentencias GO pueden evitarse por completo.
2. Si los objetivos para las sentencias GO pueden colocarse cerca del principio de un programa la ejecución será más rápida. Ello se aplica, en particular, a aquellos números de línea que se utilizarán, con frecuencia, por otras partes del programa.
3. Lo mismo que los números de línea han de encontrarse cuando se requiere, también es así con los valores de las variables. Si un número no cambia durante la ejecución de un programa, ayudará en lo que concierne a la velocidad, si no en utilización de memoria, emplearle como número y no como una variable. Por ejemplo.

   ```
   10 LET z = 0
   20 IF p > z THEN
   ```

   es marginalmente más lento que

   ```
   20 IF p > 0 THEN...
   ```

   aunque, si el número cero se utiliza mucho en el programa, la primera versión empleará menos memoria.
4. Planifique los programas de forma cuidadosa y trate de descomponerle, en la medida de lo posible, en módulos independientes. Un programa, que puede crecer como Topsy (Desbarajuste), es muy probable que se haga algo ineficaz e incoherente que duplica

algunos procesos y hace otros de una manera menos llena de intenciones (y, por consiguiente, lenta).

1.2 Programación estructurada

Pocas ediciones de las revistas de computadoras personales dejan de mencionar este tema, y de ser así, transcurrido un breve plazo de tiempo de aprendizaje de un lenguaje de computadora, la mayoría se da cuenta de que el proceso de elaborar el plan total de un programa es, al menos, tan importante como la capacidad de codificar las líneas individuales de BASIC. Los usuarios de computadoras aficionados han de tener sus propios arquitectos, delineantes y albañiles cuando construyen programas y un procedimiento estructurado es solamente una extensión de la práctica habitual adecuada. En muchos casos, el trabajo primitivo se realiza mejor utilizando un lenguaje generalizado que se denomina *lenguaje de diseño de programas o bien pseudocódigo*. Qué nombre elija carece de importancia, porque lo que se tiene realmente es una "casa a medio camino" entre el lenguaje ordinario y casi cualquier lenguaje de computadora de su elección. En el resto de este capítulo, introduciré las ideas principales de pseudocódigo (elija ese nombre solamente porque es más corto), que se utilizará, de vez en cuando, en el resto del libro para diseñar programas y partes de los mismos. A muchos puristas les gustaría una coincidencia muy estrecha entre las descripciones de pseudocódigo de programas y el propio programa real. Muy pocas, si las hay, versiones de BASIC lo permiten. El lenguaje del Spectrum no es ninguna excepción y encontraremos que hay dos tareas distintas que hacer: primero, traducir nuestros pensamientos en una descripción de pseudocódigo y luego, traducir el resultado en BASIC.

1.3 Elementos de pseudocódigo

No es mi intención colocarme por mí mismo en una "camisa de fuerza" al definir el tipo de descripciones de pseudocódigo que utilizaré en este libro. Ya tengo bastante con enfrentarme a las estrictas reglas de BASIC y puesto que pretendemos establecer un puente entre nuestro idioma y el lenguaje de la computadora que sea para nuestro propio uso (y no para el de la computadora) trataré de ser lo más flexible posible. Nadie debe sentirse obligado a seguir mi ejemplo; el pseudocódigo es una filosofía de planificación y no otra disciplina inflexible.

Las palabras y funciones de BASIC (LET, THEN, AND, INT,...) se utilizarán con libertad y se escribirán con letras mayúsculas. Los nombres de las variables se escribirán en letras minús-

culas, utilizando un signo del dólar cuando sea adecuado, pero sin ninguna restricción sobre el número de caracteres implicados. Con frecuencia, utilizará caracteres subrayados para unir las palabras individuales en un nombre de variable. La palabra PROC (por procedimiento) y una breve descripción se utilizarán para encabezar cada sección de código. Por ejemplo:

```
PROC Carga de electricidad
//cambio_unitario en peniques, cambio_fijo en libras//
LET unidades_usadas = lectura_medidor_actual-lectura_medi-
    dor prev
LET unidades_coste = INT (unidades_utilizadas * cambio_uni-
    tario)/100
LET total = cambio_fijo + unidades_coste
ENDPROC
```

La sección entre caracteres de dobles barras "//" es un comentario y este último se incluirá en donde pueda parecer de utilidad. Téngase presente una de las ideas más fundamentales en el pseudocódigo: en donde hay un principio (PROC), también hay un final (ENDPROC). El mismo principio se utiliza en el siguiente ejemplo, que supone un sistema fiscal en el que una tasa básica del 30% se grava sobre las 15.000 libras de renta imponible (el resto después de las deducciones se ha restado del salario total) y una tasa superior al 50% se aplica sobre lo que esté por encima de dicha cantidad.

```
PROC Cálculo de impuestos
// Se supone que las deducciones se han calculado ya//
LET impuesto_a_pagar = 0
LET imponible = salario_deducciones
IF imponible >15000 THEN
   LET imponible_alto = imponible – 15000
   LET impuesto_a_pagar = imponible_alto * 0,5
   LET imponible = 15000
ENDIF
LET impuesto_a_pagar = impuesto_a_pagar + impues-
to * 0,30
ENDPROC
```

Obsérvese ENDIF que, junto con el sangrado, indica claramente los procesos que han de realizarse si se cumple la condición y en dónde ha de reanudarse si no se cumple. En un programa BASIC, esto se efectúa mejor utilizando una línea de sentencia múltiple (línea 130, fig. 1.1) sin la cual se hace difícil evitar el empleo de sentencias GOTO. Aparte de cualquier consideración sobre la velocidad, estas sentencias hacen que el programa sea menos fá-

cil de leer. Obsérvese que, en la línea 140, es importante tener la variable iap a ambos lados de la asignación, para poder atender los casos en que se haya tenido la más alta tasa de cálculo.

```
 10 INPUT "salario";salario
 20 INPUT "deducciones";deducc
100 REM calculo impuesto **************
110 LET iap=0
120 LET impon=salario-deducc
130 IF impon>15000 THEN
     LET imponalt=impon-15000
      LET iap=imponalt*0.5:
       LET impon=15000
140 LET iap=iap + impon*0,3
150 PRINT "Imp a pagar es ";iap
```

Figura 1.1.

Cuando haya de utilizarse una sentencia IF, la opción de utilizar una cláusula ELSE es muy valiosa:

IF t$="Lunes" THEN LET j$="Lavado" ELSE LET j$="Planchado"

El Spectrum BASIC no tiene esta característica por lo que tenemos que abordar el problema. A veces, ello significa utilizar una sentencia GOTO:

```
10 IF d$ = "Lunes" THEN LET j$ = "Lavado" : GOTO 30
20 LET j$ = "Planchado"
30 ...
```

pero GOTO se puede evitar, con frecuencia, utilizando las funciones "lógicas", o booleanas (AND, OR, NOT), que trataré con mayor amplitud en el Capítulo 2.

```
10 LET j$ = ("Lavado" AND d$ = "Lunes") +
                         ("Planchado" AND d$ <>"Lunes")
```

En pseudocódigo, prefiero utilizar el "sangrado" (desplazamiento del margen hacia la derecha) para las cláusulas ELSE, como se muestra en el siguiente ejemplo, que calcula una calificación para un estudiante a partir de las notas obtenidas en dos tareas asignadas. Las reglas para hacerlo así se establecen en las líneas de comentarios al principio del programa. Aunque las cláusulas ELSE no puedan traducirse directamente a Sinclair BASIC, son de gran utilidad para aclarar la forma en que actúa un proceso:

```
PROC Resultados de evaluación
//El estudiante es suspendido si una u otra nota es menor que el 40% o si la media es inferior al 50% //
//Aprobado para una media del 50% o superior y con ninguna nota inferior al 40% //
//Mérito para nota igual o superior al 65% //
//Supongamos que hemos obtenido dos notas, nota1 y nota2 //
   IF nota1 < 40 OR nota2 < 40 THEN
   LET resultado $ = "Suspenso"
ELSE
   LET media = (nota1 + nota2)/2
   IF media < 50 THEN
      LET resultado $ = "Suspenso"
   ELSE
      IF media > = 65 THEN
      LET resultado $ = "Mérito"
   ELSE
         LET resultado $ = "Aprobado"
         ENDIF
      ENDIF
   ENDIF
   ENDPROC
```

la utilización de una línea de objetivo única (200, fig.1.2) y la variable r$ hace al programa BASIC (fig. 1.2) mucho menos complicado que podría haber sido de cualquier otro modo. Si el proceso de decidir un resultado es largo y la línea de objetivo está bastante alejada, puede conseguirse una mayor legibilidad utilizando:

```
30 LET líneaimpresión = 1200
```

y

```
110...: GOTO líneaimpresión, etc
```

```
 10 INPUT "notas";m1,m2
100 REM Decision de resultado *****
110 IF m1<40 OR m2<40 THEN
     LET r$="Suspenso":
     GO TO 200
120 LET media=(m1+m2)/2
130 IF media<50 THEN
     LET r$="Suspenso":
     GO TO 200
140 IF media>=65 THEN
     LET r$="Merito": GO TO 200
150 LET r$="Aprobado"
200 PRINT "El resultado es ";r$
```

Figura 1.2

1.4 Sentencia CASE

En el ejemplo anterior se tomó una decisión entre tres calificaciones posibles con el empleo de tres sentencias IF. La sentencia CASE en pseudocódigo permite una descripción clara de situaciones en donde existen muchas posibilidades y nos permite evitar programas que tengan tantos sangrados que comiencen a parecer como "serpientes" muy ágiles. Se suele reservar para dilemas en los que el curso de acción viene determinado por el valor de una sola variable, como cuando se arroja un dado, pero vale la pena que tengamos presente que el pseudocódigo es una ayuda para una buena planificación y no un conjunto fijo de reglas que exigen una estricta observancia. La notación para ello se muestra a continuación.

```
//Fecha de nacimiento indicada por b$//
CASE d$
  b$ = "Lunes"
    LET c$ = "bello rostro"
  b$ = "Martes"
    LET c$ = "lleno de gracia"
  b$ = "Miércoles"
    LET c$ = "lleno de pena"
  b$ = "Jueves"
    LET c$ = "llegará lejos"
  b$ = "Viernes"
    LET c$ = "amante y generoso"
  b$ = "Sábado"
    LET c$ = "trabaja duro para ganarse la vida"
  b$ = "Domingo"
     LET c$ = "amable y muy alegre"
ENDCASE
```

Solamente una de las alternativas ha de tener efecto. Una opción final incondicional, que abarque todo, es admisible si no se cumpliera ninguno de los casos específicos y para ello utilizaremos OTHERWISE (de cualquier otro modo). Su empleo se mostrará en el programa final de este capítulo, que es una simulación del juego de dados.

1.5 Bucles

Casi todos los programas útiles implican alguna clase de proceso de iteración en bucle. Las sentencias en el interior de un bucle se repiten hasta que se satisfaga alguna condición, después de la cual sale el programa y (normalmente) lleva a la siguiente sentencia

después del bucle. La técnica de programación más frecuentemente utilizada para la iteración en bucle es el método de FOR...NEXT, pero tiene un carácter limitador puesto que el número de limitaciones ha de darse en la primera sentencia y, en muchas ocasiones, simplemente no es posible predecir, con anticipación, cuántas veces se repetirá el proceso. Utilizará dos construcciones de pseudocódigo para describir los bucles.

```
1. WHILE <condición >
     (sentencias a repetirse)
   ENDWHILE
2. REPEAT
     (sentencias a repetirse)
   ENDREPEAT ON <condición >.
```

La única diferencia real entre las dos es la posición de la condición que ha de probarse. En la primera, está al principio por lo que las sentencias entre paréntesis no se ejecutarán en absoluto, a menos que sea verdadera la condición al comienzo. La segunda siempre da lugar a que el bucle se ejecute al menos una vez, porque la condición no se prueba hasta el final propiamente dicho. En muchas situaciones, puede utilizarse una u otra y los programadores suelen tener preferencia por un tipo u otro. Los bucles FOR...NEXT se pueden describir en una u otra forma, por ejemplo:

```
10 LET suma = Ø
20 FOR c = 1 TO 8
30 LET suma = suma + c
40 NEXT c
50 PRINT "El total de los ocho primeros enteros es suma podría
   hacerse como:
PROC Suma ocho
LET Suma = Ø
LET conteo = ØWHILE conteo < = 8
   LET conteo = conteo + 1
   LET suma = suma + conteo
ENDWHILE
PRINT...
ENDROC
```

Dejaré al lector la libertad de suministrar la versión REPEAT del pseudocódigo, pero ha de tenerse en cuenta una consideración

importante. Si el valor final en un bucle FOR...NEXT es menor que el primero (con un tamaño de paso positivo), el bucle no se introducirá en absoluto. Por ejemplo:

```
10 FOR c = 2 TO 0 STEP 1
20 LET n$ = "Sentencia a repetirse"
30 NEXT c
40 PRINT n$
```

dará lugar a un mensaje de error "variable not found" (variable no encontrada), porque el control pasará directamente desde la línea 10 a la línea 40. Si la línea 10 se modificara para tener:

```
10 FOR c = 2 TO 2 STEP1
```

no se producirá ningún error; el intérprete añade el valor de STEP al final del bucle y produce la salida del mismo cuando el conteo es más grande que el límite superior. Ello significa que una versión REPEAT de un bucle FOR...NEXT sólo es correcta si los dos límites y el tamaño del paso (STEP) son tales que aseguren que las sentencias en el interior del bucle se repitan al menos una vez.

Téngase presente que la forma en que trabajan los bucles FOR...NEXT varía de un intérprete de BASIC a otro. He descrito la interpretación más común (lo que también es cierto para el Spectrum BASIC), pero los lectores que utilicen otras máquinas deben comprobar lo que sucede cuando se ejecutan programas como los anteriores.

El siguiente programa muestra un bucle REPEAT en acción e ilustra también la observación hecha anteriormente sobre la conservación de un programa junto con sus variables. El procedimiento "Plan de ahorro" mantiene el registro de una cuenta bancaria en la que se ingresa una cantidad diferente cada mes. El tipo de interés se puede cambiar también cada vez que se haga un depósito, que solamente es realista en momentos de cambios rápidos en el ámbito económico. La parte principal del programa es un bucle simple y el pseudocódigo siguiente representa la etapa de planificación primitiva.

```
PROC Plan de ahorro
//Establecimiento de las variables necesarias//
REPEAT
  //Visualización del estado de cuenta//
  //Permitir una SALIDA del bucle aquí//
```

```
    INPUT depósito_mensual
    INPUT tipo_de_interés
    //Calcular interés//
    LET total = total + interés
  ENDREPEAT ON total > = objetivo
  //Mensaje de felicitación//
  ENDPROC
```

El listado del programa se muestra en la figura 1.3. El bucle utiliza las líneas 1000 a 1200. Obsérvese que la condición ENDREPEAT en la línea 1200 ha de utilizar if, por lo que es la condición para continuar (total < objetivo) la que se utiliza en lugar de ella para la salida.

```
100 REM Planificacion de ahorros ******
110 LET tdep=0: LET tint=0
120 LET depm=0: LET intm=0
130 LET mes=0: LET tipo=0
140 LET total=0
150 INPUT
   "Cual es su objetivo: "
   ;objet
1000 REM Repeticion bucle *****
1010 CLS : PRINT "Mes:";mes;
     AT 2,8; INVERSE 1;
   "Planificador de ahorros"
  AT 4,10;"Objetivo £";objet
1020 PRINT AT 6,0;
     "Total depositos: £";tdep
1030 PRINT AT 7,0;
     "Total intereses: £";tint
1040 PRINT AT 8,0;
     "Saldo total  : £";total
1050 PRINT AT 20,0; FLASH 1;  "'
e' para salir, cualquier otro seguir"
   : PAUSE 0:
   PRINT AT 20,0;TAB 31:
   IF INKEY$="e" THEN
GO TO 1300
1060 INPUT FLASH 1;
     "Deposito en este mes: £";
      depm
1070 PRINT AT 10,0; INVERSE 1;
     "Deposito este mes:       £"
     ;depm
1080 INPUT FLASH 1;
     "Tipo de interes este mes:";
      tipo
```

```
1090 PRINT AT 12,0; INVERSE 1;
    "Tipo de interes este mes: "
    ;tipo;"% "
1100 REM Calculos ***
1110 LET tdep=tdep + depm
1120 LET fm=tipo/(12*100):
     LET intm=tdep*fm
     LET tint=tint+intm
1130 LET total=tdep+tint
1140 PRINT AT 20,2; FLASH 1;
    "Pulse cualquier tecla para
     continuar"
1150 PAUSE 0;
     PRINT AT 20,0;TAB 31
1160 LET mes=mes+1
1200 IF total objetivo THEN
     GO TO 1000
1210 REM Hagalo ***************
1220 BEEP .2,30:BEEP .1,20
1230 CLS : PRINT AT 10,8;
     FLASH 1;"Felicidades"
1240 PRINT ''"Despues ";mes;
    " meses ha alcanzado"
    " su objetivo! ";
1250 PRINT "Su total sera "
    "total;" a final de mes."
1300 REM SAVE prog/datos ******
1310 SAVE "plan de ahorros" LINE 1000
```

Figura 1.3.

En los programas de trabajo suele ser necesario proporcionar una salida desde dentro del bucle; en este caso, para permitir un retardo (de un mes) entre depósitos. La salida se proporciona en la línea **1050** y va a las líneas finales del programa y no al final del bucle. Cuando ello sucede, se visualiza el mensaje habitual (arrancar cinta y luego, pulsar cualquier tecla), por lo que si la grabadora de cinta está adecuadamente puesta a punto, el programa se grabará en la cinta de tal forma que, cuando se recargue, el control pasa al comienzo del bucle y se da el estado de cuenta del último mes. Debe ser posible utilizar este programa durante un período de tiempo considerable para mantener una verificación automática sobre una cuenta o puede jugar a “¿qué pasaría si...?” ejecutándole, en un avance de cinco años, en una sóla sesión. Circunstancialmente, las líneas **100** a **150** podrían suprimirse después de que el programa se haya ejecutado (RUN) una sola vez sin perderlos valores almacenados en las variables, aunque después de su inserción en el programa y de su borrado sea necesario teclear GOTO **1000** y no RUN.

1.6 Juego de dados

El último programa del capítulo reúne la mayor parte de las técnicas de pseudocódigo mostradas. Se trata de un juego de apuestas, aunque he dejado la provisión de dicho aspecto al lector (ganar o perder es una cuestión de pura suerte, en cualquier caso). El jugador tira dos dados. Si la puntuación combinada es 7 ó 11, gana inmediatamente, pero un total de 2, 3 ó 12 significa una pérdida categórica. Cualquier otro valor se hace el "punto" del jugador y debe continuar tirando los dados hasta que vuelva a lograr su "punto" (ganando) o que obtenga 7 (perdiendo). Si el juego llega más allá de la primera etapa, puede ser bastante emocionante si, tirada tras tirada, no se obtiene el resultado buscado. A continuación, damos la versión de pseudocódigo del programa.

```
PROC juego de dados
//Tirar los dados y calcular la puntuación//
CASE de puntuación
  puntuación = 7 ó 11
    LET resultado$ = "Gana"
  puntuación = 2, 3 ó 12
    LET resultado$ = "Pierde"
  OTHERWISE
    LET punto = puntuación
    REPEAT
      //Tirar los dados de nuevo//
  IF puntuación = 7 THEN
      LET resultado$ = "Pierde"
  ELSE
    IF puntuación = punto THEN
      LET resultado$ = "Gana"
    ENDIF
  ENDIF
  ENDREPEAT ON resultado$ = "Gana" o "Pierde"
ENDCASE
ENDPROC
```

En el programa (figura 1.4), la sección que simula la tirada de los dados se coloca como una subrutina al final del programa (línea 1000) y utiliza la función RND dos veces. No sería correcto emplear

```
LET puntuación = 1 + INT (12 * END)
```

puesto que esto hace que todos los totales sean igualmente probables y el juego se basa en el hecho de que cuando se tiran los dados, las puntuaciones en la parte media de la gama de posibilidades se obtendrán con más frecuencia que las extremas. Las con-

```
100 REM Juego de dados ********
110 CLS : LET tiradas=0:
    RANDOMIZE
120 FO SUB 1000:
    IF punt=7 OR punt=11 THEN
    LET r$="Gana": GO TO 500
130 IF punt=2 OR punt=3
    OR punt=12 THEN
    LET r$="Pierde": GO TO 500
140 LET punto=punt:
    PRINT "el punto es ";punto
150 GO SUB 1000: IF punt=7
    THEN LET r$="Pierde":
     GO TO 500
160 IF punt=punto THEN
    LET r$="Gana"; GO TO 500
170 GO TO 150
500 REM Juego acabado *********
510 PRINT r$; en ";tiradas;
   "tiradas."
520 INPUT "Otro juego (s/n)";a$
   : IF a$="s" OR a$="S" THEN
   GO TO 100
530 STOP
1000 REM Tirada ***************
1010 LET t1=1+INT (6*RND):
     LET t2=2+INT (6*RND)
1020 LET punt=t1+t2:
     PRINT punt
1030 LET tiradas=tiradas+1: RETURN
```

Figura 1.4.

diciones de CASE han de encontrarse en las líneas 120, 130 y 140. (siendo la última la opción de OTHERWISE) y el bucle REPEAT ocupa las líneas 150 a 170. Obsérvese que aunque el programa no sigue exactamente la descripción de pseudocódigo en este punto, el efecto es el mismo.

Es posible utilizar este programa para investigar la equidad del juego de datos. Un bucle FOR...NEXT colocado alrededor de las líneas 100 a 510 puede emplearse para jugar el juego, de forma automática, cualquier número deseado de veces. En una ejecución. 1000 juegos dieron 527 resultados de ganar con una media de 3,34 tiradas por juego, por lo que resulta evidente que este juego es muy equilibrado. Cualquier elemento de habilidad en el jugador radica en calcular las desigualdades y en ajustar la apuesta después de que la primera tirada no haya obtenido un resultado.

2 IF

"Si puede, llene el minuto inexorable con un valor real de sesenta segundos a distancia".

Rudyard Kipling

Soy un ávido lector de las revistas de computadoras que han "florecido" en nuestros quioscos en los últimos años: Uno de mis más "sádicos" placeres, obtenidos de dichas publicaciones, es una colección de "disparates" de programación. Por supuesto, la mayor parte de los programas impresos son excelentes ejemplos de ingenuidad y un estudio concienzudo de los esfuerzos de otros puede ser muy conveniente. Sin embargo, de vez en cuando, ejemplos malos se deslizan a través de la red de criba. En la mayor parte de las ocasiones son simplemente consecuencia de una deficiente planificación, con construcciones divagantes que han de ser puestas en orden (¡Y quién podría arrojar la primera piedra!), pero, a veces, se trata de elementos de codificación que no pueden hacer lo que pretendía quien lo escribió. Muchos de ellos comienzan con IF.

En esta etapa inicial de nuestro avance en las actividades de cómputo, la mayor parte de las máquinas y de los lenguajes son sirvientes implacables, tratando de hacer, con interés, obstinación y exactitud, lo que nuestros programas les piden, prescindiendo de lo que pretendíamos exponer. Por supuesto, los errores de sintaxis son objeto de informe, pero las "pifias" lógicas pueden quedar ocultas y sólo se detectan en el momento más inoportuno. Cuanto más complejo es el programa tanto más profundamente "atrincherado" es probable que esté el error, por lo que no puede sorprendernos encontrarles en las publicaciones impresas de vez en cuando. Mi colección de disparates tiene varios ejemplos semejantes a:

```
100 IF a$ = "p" THEN LETsp = 10: IFa$= "q"THEN
LET sp = 20
```

El resultado de esta línea de programa es que si a$ = "p", a la variable sp se le dará el valor de 10; sin embargo, si a$ = "q", a sp no se le afectará con nada y mantendrá su valor anterior, si lo tuviera. Es muy fácil cometer esta clase de error, sobre todo cuando el intér-

prete de Sinclair no tenga una cláusula ELSE. Lo que debe recordar es que ENDIF está siempre al final de la línea de programa. Dicho de otro modo, cuando se encuentra una sentencia IF, solamente una de dos cosas puede suceder: si la condición es verdadera, se "acatará" el resto de la línea después de la sentencia THEN; sin embargo, si la condición no es verdadera, el control pasa al siguiente número de línea y no a las sentencias separadas de la primera por dos puntos. A este respecto, una línea constituida por múltiples sentencias separadas por dos puntos pueden dar lugar a un comportamiento diferente al que tendrían las mismas sentencias en líneas individualmente numeradas. En el ejemplo anterior, la segunda IF nunca se alcanza a no ser que tenga a$ = "p" y, en ese caso, no hay nada que probar para ver si a$ es igual a "q". Una versión correcta de lo que el programador pretendía es:

```
100 IF a$ = "p" THEN LET sp = 10
110 IF a$ = "q" THEN LET sp = 20
```

pero veremos más adelante que, a veces, hay métodos más cómodos.

IF...THEN es probablemente la más difícil de todas las construcciones de BASIC para su utilización efectiva y es una cuestión de estilo individual de cómo responder al problema. En el resto de este breve capítulo, trataré de demostrar algunas formas de cómo abordar el problema. Naturalmente, corresponde a los lectores la decisión de adoptar el estilo del autor, de forma total o parcial, pero la comprensión de las técnicas utilizadas será de utilidad en la interpretación de los programas en el resto del libro.

2.1 Bifurcación

IF se suele utilizar como un medio de dirigir el flujo del programa, o su control, a lo largo de un camino diferente a partir del principal, lo mismo que la línea de bifurcación de una vía férrea sale de la línea principal. Si se utiliza de esta forma, es importante recordar que se trata de un conmutador de dos vías: el control va a lo largo de la bifurcación o continúa, en sentido descendente, a través de la línea principal, como en la figura 2.1. En tal caso, la línea principal suele ser un nuevo punto de reunión más adelante y, entonces, el empleo de las líneas de multisentencias en la bifurcación puede ser de gran utilidad (ver, por ejemplo, el programa de cálculo de los impuestos en el capítulo 1). Si queremos que el punto de reunión en la línea principal sea diferente a aquel en donde se produjo la bifurcación, la mejor forma de conseguirlo es utilizando una GOTO al final de la bifurcación (no hay ningún límite para la longitud de una línea en Sinclair BASIC). Este método se utilizó en el programa del juego de dados al final del último capítulo:

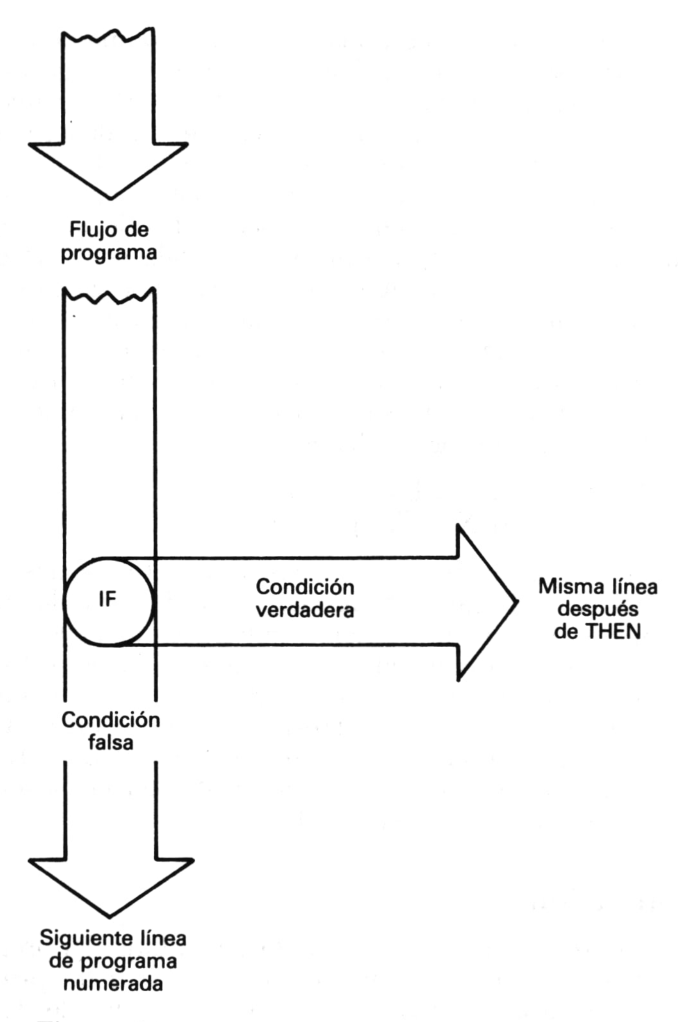


Figura 2.1.

```
150 GOSUB 1000:IF puntuación=7 THEN LET r$ "pierde":
GOTO 500
160 ...
```

ALTERNATIVAS A IF

Cuando las condiciones para la bifurcación (o decisión sobre otra acción) se hacen complicadas, la falta de una cláusula ELSE, junto con el hecho de que ENDIF se tome como estando al final de una línea de programa, contribuyen a complicarnos la vida. Las funciones lógicas, o booleanas, pueden proporcionar, con frecuencia, una alterna-

tiva muy satisfactoria que, a veces, será de ejecución más sencilla. La lógica booleana de los valores de la verdad para expresiones tales como:

está lloviendo
el valor de x es 5
y es un múltiplo de 32
esta es la segunda tirada y la puntuación es 7

En los libros sobre lógica, los "valores" asignados a dichas expresiones son verdadero o falso, dependiendo de si, en algún caso particular, las sentencias sean verdaderas o falsas. En Sinclair BASIC, se pueden evaluar las variables adecuadas correspondientes para codificar las sentencias (siendo los resultados "uno" o "cero" según sean verdaderas o falsas respectivamente). Por lo tanto:

```
100 LET t = (x = 5)
```

dará lugar a que t tenga el valor "uno" si x es igual a 5 y "cero" si no lo es (los paréntesis no son estrictamente necesarios pero nos ayudan a "explorar" la expresión). Por ello, la línea del programa es equivalente a poner:

```
100 IF x = 5 THEN LET t = 1
110 IF x <> 5 THEN LET t = 0
```

Quizás sea útil considerar que BASIC evalúa la expresión x = 5 en exactamente la misma forma que calcula el valor de x + 5, o incluso 7*6, pero solamente permite dos resultados posible. Ciertamente, puede tratar expresiones booleanas en exactamente la misma forma que las funciones aritméticas más frecuentemente encontradas. Muchas versiones de BASIC permiten, ahora, dichas operaciones, pero ha de tenerse presente que no todas dan las mismas respuestas. Aunque todas ellas dan el resultado cero para "falso", en muchas versiones de BASIC la condición de "verdadero" corresponde a un valor de −1, lo que puede parecer extraño, pero es, de hecho, el resultado de una comparación "bit a bit", que utiliza las versiones binarias de los números implicados y efectúa la comparación dígito a dígito. Todas las relaciones en Sinclair BASIC puede utilizarse en las operaciones booleanas, por lo que:

```
100 LET t1 = (x > 5)
110 LET t2 = (x < = 32)
120 LET t3 = (x = 5 *INT(x/5))
130 LET t4 = (x > 0 AND x < 31)
```

son todas ellas líneas del programa perfectamente válidas que prueban si x satisface una condición y pone la variable t a "uno" o

a "cero". La tercera determina si x es un número entero divisible por 5; la última, si x está entre 0 y 31. Las pruebas se hacen todavía más útiles si, por ejemplo, utilizamos:

```
100 LET x = x* (x > 0)
```

Ello tiene exactamente el mismo efecto que:

```
100 IF x < 0 THEN LET x = 0
```

pero no hay ninguna necesidad con la primera versión (booleana) de que nos preocupemos con el problema de ENDIF (el objeto de la sentencia IF implicada se indica por los paréntesis).

Las funciones booleanas actúan lo mismo que con las variables de cadena en el lado derecho de la asignación, por lo que en lugar del primer ejemplo en este capítulo:

```
100 IF a$ = "p" THEN LET sp = 10
110 IF a$ = "q" THEN LET sp = 20
```

podríamos utilizar también:

```
100 LET sp = 10* (a$ = "p") + 20* (a$ ="q")
```

la única diferencia radica en que, en el primer caso, en valor antiguo de sp se mantendrá invariable si a$ no es "p" ni "q", mientras que la versión booleana hará que la variable sp se ponga a "cero". Si fuera necesario mantener el valor antiguo, sería posible utilizar:

```
100 LET sp = sp + (10-sp)*(a$ = "p") + (20 - sp)*(a$ = "q")
```

pero quizás habría muy pocas circunstancias en las que querríamos emplear dicha expresión emplicada. La única vez que el autor ha utilizado dicho mostruo fue en medio de una función definida por el usuario, en donde no están permitidas las condiciones IF.

2.2 Bifurcación con expresiones booleanas

Hay muchas ocasiones en las que más de dos alternativas se requieren en un punto de decisión: una construcción de bloques (CASE). Por ejemplo, cuando al usuario de un programa se le requiere para dar una respuesta a una pregunta en la pantalla, podemos encontrarnos con el empleo de una sección de programación similar a:

```
100 INPUT "Tiene el animal cuatro patas?:",a$
110 IF a$ = "sí" THEN GOTO 500
120 IF a$ = "no" THEN GOTO 600
130 GOTO 100
```

```
500 REM Se trata de un animal de cuatro patas
510 ...
600 REM Se trata de animal de dos patas (¿no?)
```

que salva cualquier eventualidad volviendo a repetir la pregunta si la respuesta es incorrecta. Es más claro, en tales circunstancias, emplear:

```
100 INPUT "Tiene el animal cuatro patas?:"a$
110 GOTO 100 + 400 * (a$ = "sí") + 500 * (a$ = "no")
```

que, aunque no se comprenda tan fácilmente, tiene exactamente el mismo efecto, utiliza menos memoria y probablemente se ejecuta con más rapidez. Las expresiones entre paréntesis serán objeto de evaluación individual, produciendo "uno" o "cero" pero no dos "unos" a la vez, puesto que a$ no puede ser igual a la vez a "sí" y a "no", y el resultado de la expresión después de GOTO será 100, 500 ó 600. Sinclair BASIC no tiene ninguna objección en calcular el número de línea de un objetivo para un GOTO o GOSUB; en realidad, parece no tener ninguna objeción en calcular cualquier valor requerido en un programa, en tanto que el resultado sea adecuado. Sin embargo, obsérvese que la desviación ("offset") de 100 (la primera parte de la expresión calculada) es necesaria, pues, sin dicho valor, el control podría dirigirse a la línea 0 (en efecto, el principio del programa). Un GOSUB no podría actuar en exactamente la misma manera, pero podríamos aprovecharnos de la subrutina adicional requerida con algo similar a lo siguiente:

```
100 INPUT "Tiene el animal cuatro patas?:";a$
110 LET err = 0:
    GOSUB 200 + 300 * (a$ = "sí") + 400 * (a$ = "no"):
    IF err THEN GOTO 100
120 REM Continuar con programa
200 PRINT "Respuesta incorrecta": LET err = 1: RETURN
```

En este caso, la variable err se utiliza como una pseudovariable. Señaliza que se ha producido un error, de modo que, después del retorno desde la subrutina, el programa puede utilizar su valor para decidir qué hacer a continuación. No hay ninguna necesidad de poner IF err = 1 THEN ..., pues "uno" es, de nuevo, lo mismo que "verdadero" en el contexto. Si se requieren varias de dichas respuestas en el curso del programa, la subrutina en la línea 200 podría utilizarse como el objetivo cada vez que se da una respuesta incorrecta, puesto que el mecanismo de RETURN encuentra siempre su forma de volver al lugar correcto, incluso si está en la parte media de una línea.

2.3 Cadenas

Todos los cálculos realizados hasta ahora han implicado variables de números. Sólo hay una función del tipo booleano que puede utilizarse con cadenas. De forma estricta, no se trata de una función lógica, pero actúa de una forma muy similar.
Podemos utilizar:

```
100 LET v$ = ("con cuatro patas" AND a$ = "sí")
    + ("con dos patas" AND a$ = "no")
```

en lugar de:

```
100 IF a$ = "sí" THEN LET v$ = "con cuatro patas"
110 IF a$ = "no" THEN LET v$ = "con dos patas"
```

en tanto que no tengamos necesidad de recordar el valor anterior de v$; si a$ no es "sí" ni "no", v$ se hará una cadena nula o vacía por la primera versión.

En este capítulo he intentado presentar algunas alternativas, que son métodos de dirigir el control en un programa y de asignar valores a variables sin utilizar la construcción de IF. Por supuesto, no pretendo que siempre sean adecuados y admito que hasta que no se domine la función booleana, no familiar, puede parecer menos asequible. Sin embargo, siempre es útil ser diestro en muchas cosas diferentes y las versiones alternativas se utilizan mucho en el resto de este libro.

3 INDEXACION Y BUSQUEDA DE FICHEROS

En el ámbito de las computadoras, el fichero de palabras abarca una amplia gama. Hay ficheros de visualización, ficheros de programas, ficheros de impresión, ficheros de datos, ficheros de órdenes, ficheros de textos, etc. De hecho, prácticamente cualquier colección de información útil se cubre por este término, y cualquier definición que se intentara sería tan amplia que, casi con absoluta certeza, sería carente de utilidad. Afortunadamente, en este libro sí estamos interesados por una gama de ficheros menos amplia constituida por aquellos que retienen información de la clase encontrada, por ejemplo, en una guía telefónica. "Nuestros" ficheros retendrán información relativa a la vida diaria más bien que con respecto a las operaciones de la computadora y estarán, en conjunto, bastante estrictamente organizados, por lo que podremos extraer cualquier elemento de datos particular con bastante rapidez.

3.1 Almacenamiento

En el momento de la escritura, la única forma en que el sistema de Sinclair permite el almacenamiento permanente de ficheros es en cinta de casete; pueden almacenarse en la memoria de la computadora mientras esté aplicada la alimentación, pero se pierden tan pronto como se corta dicha alimentación. Puesto que el sistema operativo para casete no permite la transferencia automática de información desde la cinta bajo control del programa, nos veremos limitados a ficheros de tamaño bastante pequeño para poder ser admitidos en la memoria interna. Si dispone de una máquina de 48K, sin embargo, encontrará que hay suficiente espacio para admitir muchos ficheros personales (mi agenda cabrá, dejando espacio de reserva, como también un catálogo de todos los ficheros de computadora de todas clases, que he utilizado en los últimos años). Con métodos más sofisticados de almacenamiento externo se puede retener una mayor cantidad pero, en cualquier caso, las técnicas empleadas son igualmente aplicables.

El Spectrum tiene un sistema operativo especialmente adecuado desde este punto de vista. La memoria interna se utiliza para almacenar el programa y las variables en un solo bloque grande, que

se transfiere a cinta de casete en masa, por lo que un programa para retener un fichero puede conservarse (SAVE) con toda la información introducida y más tarde, se cargará conjuntamente.

3.2 ¡Nunca teclee RUN!

Cuando un programa está en ejecución (RUN), se destruyen, de forma irrevocable, todas las variables en memoria (no, por supuesto, en la cinta de casete). Ello presenta un riesgo que se puede hacer mínimo de varias formas:

1. Utilice siempre la opción SAVE...LINE XXXX. Cuando un programa así conservado se carga más tarde se reiniciará, de forma automática, en donde se dejó, con toda la información intacta.
2. Escriba la instrucción SAVE (con su LINE) en el programa, de modo que se pueda ejecutar automáticamente el final de cada sesión. Los programas posteriores indicarán exactamente como ello se hace.
3. Olvide en donde está la tecla RUN y emplee siempre GOTO XXXX (XXXX significa un número de línea en el programa). GOTO tiene el mismo efecto que RUN pero no hace que se destruyan las variables. Si debe cambiarse el valor de una variable porque fuera de carácter erróneo durante el procesamiento, utilice LET (sin ningún número de línea) para su restauración. Recuerde que en el Spectrum BASIC no hay prácticamente ninguna diferencia entre las sentencias emitidas con un número de línea y las que no lo tienen.

3.3 Estructura del fichero de datos

Un fichero de datos es una forma natural de organizar la información. Solamente la terminología utilizada en los sistemas de computadoras puede necesitar explicación y, por supuesto, la forma en que los ficheros de datos son producidos por programas. En una agenda de direcciones (que constituyen un fichero), encontramos secciones independientes, teniendo cada una un nombre y una dirección correspondiente a una persona o familia. Esto es un registro y un fichero es un conjunto de registros. Dentro de cada registro utilizamos líneas para contener elementos diferentes: nombre, primera línea de dirección, segunda línea, número de teléfono, etc. En el caso de las computadoras, cada sección se denomina un campo. El conjunto de campos constituye un registro. A su vez, dentro de cada campo encontramos letras individuales, dígitos, signos de puntuación y, por supuesto, espacios. Cada uno se deno-

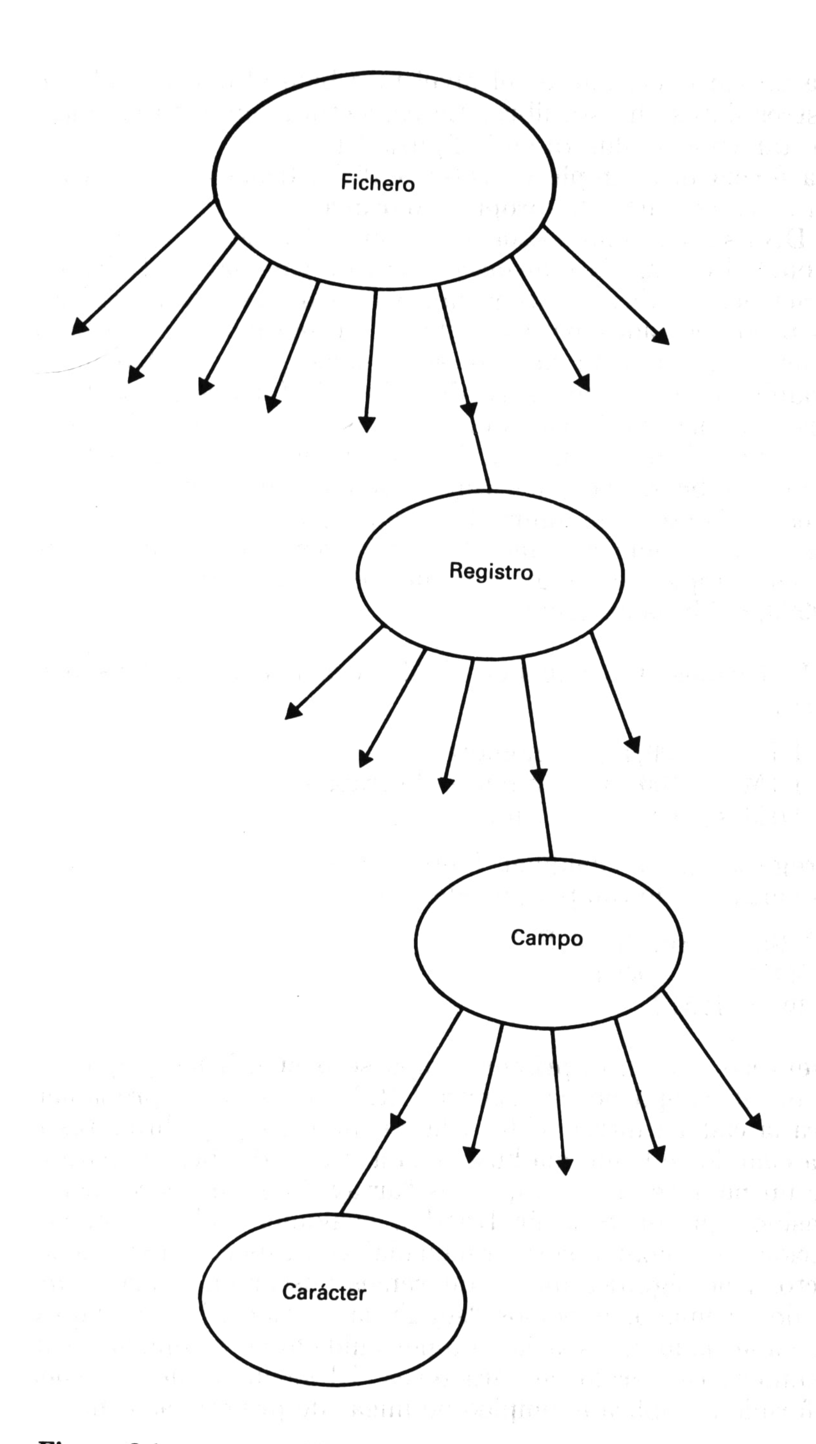


Figura 3.1.

mina un carácter, que es el nivel más bajo al que consideraremos nuestros datos en este libro. La estructura completa es una jerarquía, tal como se ilustra en la figura 3.1.

La forma más simple de retener dicho fichero en una computadora es como parte del propio programa, con el empleo de sentencias DATA, pero ello no suele ser muy adecuado si no hay ninguna posibilidad de que la información vaya a cambiarse. Todas las modificaciones y adiciones tendrían que realizarse utilizando los medios proporcionados para escribir un programa. No obstante, en ocasiones, cuando un fichero es pequeño y de cambio muy poco probable, este método será de utilidad. La mayor parte de los datos se almacenarán como el contenido de variables. Examinemos tres formas de almacenar un fichero simple que contenga un centenar de nombres, cada uno con un número de trabajo y el salario correspondiente (ver figura 3.2). Supongamos que para el nombre necesitemos, como máximo, 15 caracteres, que el número de trabajo es siempre de 6 dígitos y que todos los salarios son inferiores a 99999,99 libras esterlinas.

1. Podríamos descomponer el almacenamiento en tres matrices ("arrays")

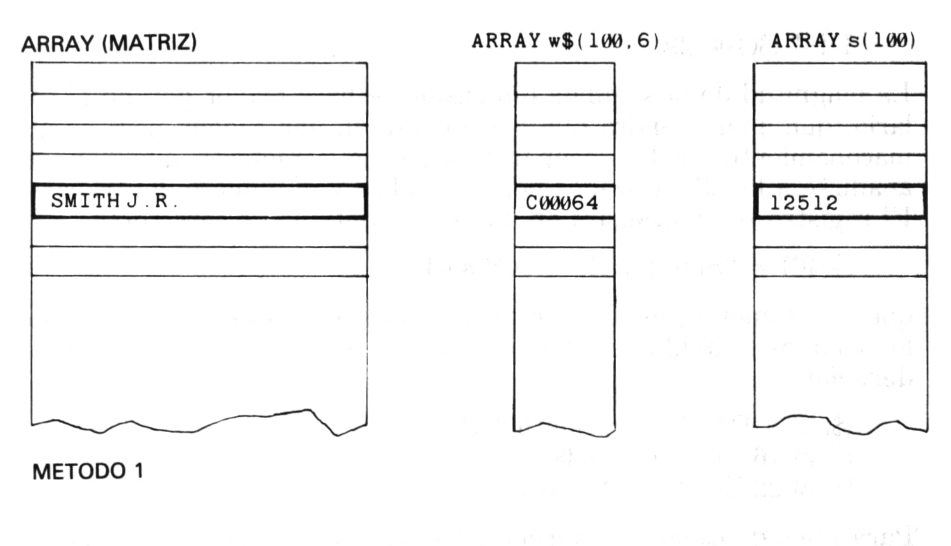
```
DIM n$(100,15)     nombres
DIM w$(100,6)      número de trabajo
DIM s(100)         salarios
```

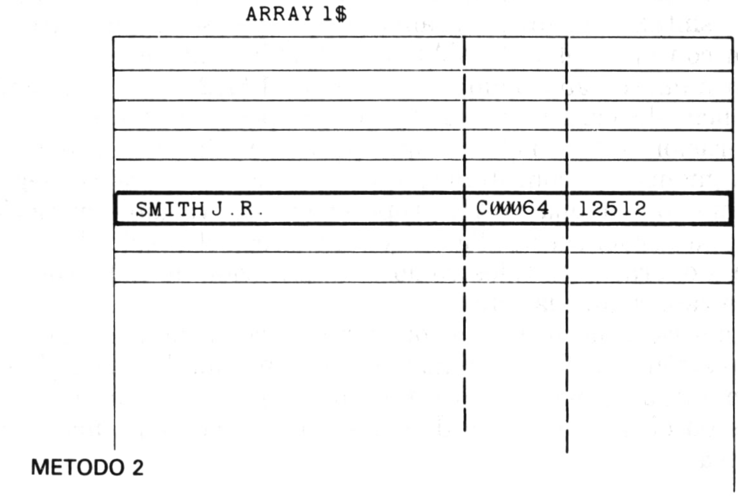
El registro para cualquier individuo particular se encontrará en tres variables de campo, por ejemplo:

```
n$(40) = "Smith a J.R."
w$(40) = "C00064"
s(40) = 12512
```

Obsérvese que en el primer campo se desperdician y que el nombre en el campo no es "Smith J.R.", aún cuando probablemente fuera la cadena introducida. Ello puede causar quebraderos de cabeza cuando se haga una búsqueda a través del fichero para encontrar un nombre. Puesto que los "arrays" de caracteres son de dimensión fijas en Sinclair BASIC, cualquier cadena con longitud inferior a la máxima será "rellenada" con espacios a su tamaño correcto. Ello significa que si queremos buscar en el array un nombre determinado, debemos "seguir la corriente" añadiendo el número adecuado de espacios o tener cuidado en examinar solamente el número requerido de caracteres. El segundo método suele ser preferible e implica el empleo de líneas de programas como:

```
100 IF n$(c,TO 9) = "Smith J.R." THEN...
```





l$

BROWN P.W.*D52134*9263*SMITH J.R.   *C00064*12512*

METODO 3

Figura 3.2.— Métodos para organizar ficheros.

2. Podría utilizarse un array de caracteres de dos dimensiones:

DIM l$(100,29)

La magnitud de la segunda dimensión se hace mayor porque el salario, que ahora tendrá que convertirse en una cadena para su almacenamiento, podría ocupar hasta ocho caracteres que han de añadirse a los 21 utilizados por el nombre y el número del trabajo. El registro se almacenará en una serie continua de caracteres.

l$(40) = "Smith J.R.   C00064125 12   "

que, ciertamente, es una forma todavía más confusa. Para extraer los campos individuales, tendríamos que utilizar una forma de indexación:

l$(40,1 TO 15) = "Smith J.R.   "
l$(40,16 TO 21) = "C00064"
l$(40,22 TO 29) = "12512   "

Para fines de cálculo, la forma del salario antes dada daría lugar a errores de sintaxis en un programa por lo que sería necesario su conversión con el empleo de VAL l$ (40,22 TO 29), con lo que se obtiene un número "adecuado". La cadena "12512   " es solamente una secuencia de caracteres en lo que respecta al BASIC y las únicas operaciones "aritméticas" que pueden realizarse en ella son las de fragmentar y concatenar, que sólo sirven para conseguir más variables de cadena. VAL proporciona una variable numérica a partir de la secuencia de dígitos y, entonces, el resultado puede someterse a operaciones tales como multiplicación de números, extracción de raíz cuadrada, etc.
La forma matricial de organización tiende a requerir más esfuerzo de programación y utiliza poco más espacio, porque la forma de cadena del número ocupa ocho octetos en comparación con los cinco requeridos en el método 1, en donde se almacena como una variable numérica.

3. Una matriz de caracteres de una sola dimensión (o una cadena simple que se expande cuando se añaden más registros). Aquí podría ahorrarse espacio separando los campos con un carácter especial:

   l$ = "........*Smith J.R.*C00064*12512*........"

   suponiendo siempre que el asterisco nunca se utilizaría como parte de, por ejemplo, un número de referencia del trabajo. Esta forma tiene el atractivo de que se ahorra espacio, incluso permitiendo los tres separadores requeridos en cada registro, pero es evidente que encontrar un registro en medio de una

cadena larga llevará un tiempo considerable, incluso en términos de la computadora y el programa tendrá que examinar cada carácter, de forma individual, para determinar si se trata de un asterisco o de simplemente una parte del registro. Volveré a la cuestión aparentemente inocente de la elección entre cadenas simples y las de una sola dimensión en el siguiente capítulo, con conclusiones sorprendentes.

De los tres métodos, mi favorito, quizás sorprendentemente, es el segundo. En primer lugar, se corresponde con la forma en que yo considero los ficheros y que no es otra cosa que una colección de datos de información. Esta era la imagen que tenía en mente cuando se concibió PROFILE y el objetivo perseguido era traducirle en una imagen real en la pantalla. Es todavía más importante el hecho de que cuando se consideran todas las operaciones necesarias (borrar, añadir, modificar, visualizar, etc.) se hace realmente más fácil de programar que el primero, que apenas se utilizará en este libro. El trabajar con el segundo tipo de fichero requerirá más experiencia en la fragmentación e indexación en cadenas pero espero que los lectores aceptarán este reto. Ello significa también que los programas mostrados más adelante en el libro podrán adaptarse, con más facilidad a las exigencias individuales.

3.4 "Bloc de notas"

El tercer método sólo se utilizará para un programa principal (en el capítulo 6), pero la aplicación de la idea se hace en la figura 3.3, que muestra el programa anterior encima de su visualización en pantalla. En este programa, una lista de artículos se almacena en una cadena de extensión simple. Cada registro tiene solamente un campo, de longitud variable, por lo que el separador de campos separa también registros. En lugar de utilizar uno de los caracteres imprimibles para esta función, se utiliza un código de control, CHR$ 13. Este es el carácter producido por la tecla ENTER y es "seguro" porque nunca se producirá como parte de algo que se teclee. En la línea 10 se inicializa la cadena. La línea 200 es un bucle pequeño; la primera sentencia busca un artículo de datos y luego, mientras se teclea algo, la segunda sentencia concatena la entrada (s$) y CHR$ 13 a la cadena 1$. La tercera sentencia de la línea 200 vuelve a la búsqueda de otra. Cuando se concluye la introducción (basta pulsar la tecla ENTER en respuesta al mensaje de solicitud de datos), el control vuelve a la línea 20. La parte media del programa imprime los artículos uno a uno. Si quiere conservar (SAVE) su lista de compras, utilice SAVE "bloc de notas" LINE 15.

```
5 REM Block de notas ****
10 LET coteo=0: LET l$="":
   LET s$=CHR$ 13
15 GO SUB 200
20 LET ll=LEN l$
100 LET st=1
110 IF st>=ll THEN STOP
120 LET i=st+1
130 IF l$(i)<>s$ THEN
    LET i=i+1: GO TO 130
140 LET fi=i-1:
    PRINT l$(st TO fi):
    LET st=fi+2: GO TO 110
200 INPUT "Articulo sig.";n$:
    IF n$<>"" THEN
    LET l$=l$+n$+s$: GO TO 200
210 RETURN
```

```
1 pan grande
Compota de fresa
Detergente liquido
Patatas
Lata grande de tomates
Lata pequena de peras
Compota de albaricoque
Bacon en lonchas
```

Figura 3.3.

Para enviar la salida a una impresora cuando se ha detenido el programa, basta teclear LPRINT 1$ (¿por qué actúa?).

Veamos cómo actúa la sección control del programa. Las variables utilizadas son:

st comienzo de un artículo (posición del primer carácter)
i desplazamiento del índice en la cadena
fi final de un artículo (último carácter antes de CHR$ 13).

En la figura 3.4 se muestra el estado de la ejecución después de que se haya impreso "detergente líquido". La variable fi apunta al último carácter impreso y st a la primera letra de "potatoes" (patatas). El índice se desplaza gradualmente a lo largo de la cadena por la línea 130 hasta que encuentre un separador (s$). Cuando lo hace, el puntero fi se desplaza (línea 140) y se imprime 1$ (st TO fi)–"potatoes". Después de una comprobación de que no se ha al-

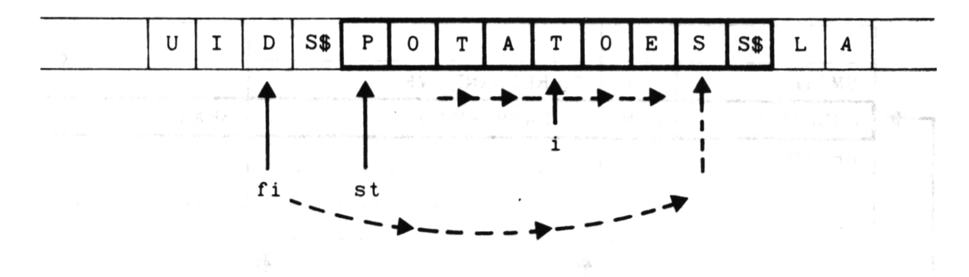


Figura 3.4.

canzado el final de la cadena (línea 11Ø), st se desplaza y se repite el ciclo. Este tipo de procedimiento se encontrará en muchos programas en este libro.

3.5 Indexación

Esta idea es fundamental para prácticamente todos los programas contenidos en este libro. En términos generales, significa utilizar una variable como un puntero para encontrar o utilizar algo más. La variable de índice suele quedar oculta a la vista de la persona que utiliza el programa; su valor no queda visible, pero sus efectos son todos ellos importantes. En el programa de "bloc de notas", todas las variables en la rutina de impresión (st, fi y i) son valores de índice, que apuntan a las partes de la cadena de datos 1$ y control que se visualiza. Hablamos de indexación en una cadena, programa o tabla o cualquier otro lugar.

En la figura 3.5. se muestra cómo indexaremos en una matriz de cabeza bidimensional. Las tres variables de índice se combinan para aislar una sección del fichero; si se cambia el valor de r, la expresión indicará el tercer campo de otro registro y si se cambia rl y rr se examinará otro campo en el mismo registro. La indexación es una técnica muy poderosa y daré algunos ejemplos de su uso, que, en su mayoría, se desarrollarán más adelante en este libro.

INDEXACION EN UNA LISTA DE SENTENCIAS DE DATA—LISTA DE VINOS

En BASIC, las sentencias READ son objeto de control utilizando un puntero, que siempre indica el siguiente artículo, o elemento de datos, que ha de utilizarse. La orden RESTORE puede utilizarse en un programa para cambiar el número de línea indicado por el puntero de datos. Veamos un programa sencillo concebido para

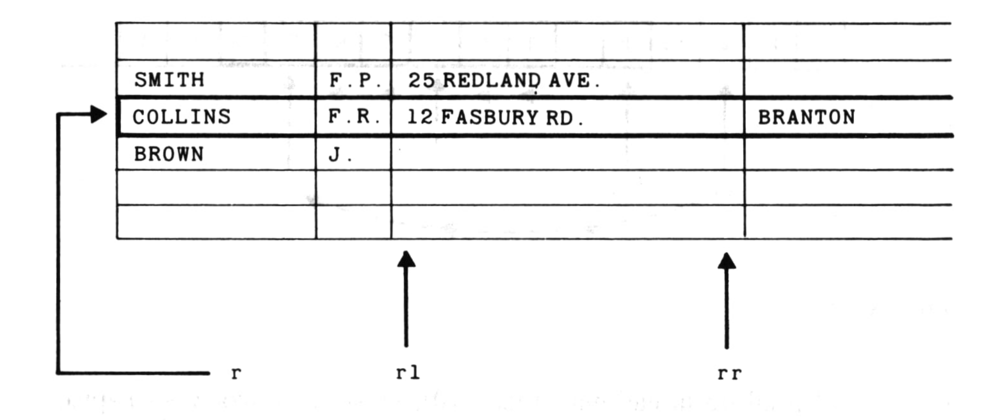


Figura 3.5.

presentar el usuario una elección de vinos en la pantalla. En el programa, los nombres y los precios de los vinos se mantienen en la forma de DATA y en la línea 110 (figura 3.6) el puntero de datos se activa para la primera sentencia DATA, que contiene el número de vinos en el menú. En la línea 120, el programa utiliza la variable de bucle i cómo un índice en la lista, con la lectura y visualización de forma sucesiva. El número de la sección de la bodega se obtiene, entonces, del usuario, se comprueba su validez y se utiliza en la sentencia, RESTORE en la línea 150, que utiliza la variable ch para una nueva indexación en la lista y, en esta ocasión, con la visualización del precio en la parte inferior de la pantalla y destacando la botella elegida con su reimpresión en el modo de intermitencia (FLASH). Ello sólo es posible porque los vinos se han colocado en sentencias DATA en líneas separadas. El programa da al usuario una nueva posibilidad de pensar, que es por lo que fue necesario incluir la primera sentencia RESTORE.

INDEXACIÓN EN UN PROGRAMA

La utilización hecha de un valor calculado para el número de línea en la sentencia RESTORE anterior puede hacerse "en paralelo" en las órdenes GOTO y GOSUB. Muchas versiones de BASIC reconocen una sentencia de la forma:

```
10 ON × GOTO 1000,1500,2000,3000
```

que transfiere el control del programa a uno de los números de línea dados, con el empleo de valor de × (1,2,3 ó 4) como un índice

```
 10 REM Lista de vinos *******
100 CLS : PRINT AT 1,12:
    INVERSE 1:"Lista de vinos"
110 RESTORE 300: READ nw
120 FOR i=1 TO nw:READ w$,p$:
    PRINT AT i+3,0;"Bin No ";
    i;"-";AT i+3,11;
  w$: NEXT i
130 PRINT AT 20,2: FLASH 1;
 "Introduzca numero de bodega";
140 INPUT ch: IF ch<>INT ch
     OR ch<1 OR ch>nw THEN
      BEEP .2,40: BEEP .1,10:
       GO TO 140
150 PRINT AT 20,0;TAB 31;:
     RESTORE 300+ch: READ w$; p$
160 PRINT AT ch+3,11;
     FLASH 1;w$
170 PRINT AT 20,2; FLASH 1;
  "  El precio es £";"  ";
  AT 21,7;"elige otro (s/n)?";
180 PAUSE 0 : IF INKEY$="s" OR
    INKEY$="S" THEN GO TO 100
190 CLS : PRINT AT 10,0;
  "Gracias, espero que disfrute"
  ;AT 11,0;"su botella de ";w$
300 DATA 14
301 DATA "Chianti Classico","3.
    10"
302 DATA "Frascati","3.28"
303 DATA "Soave","2.48"
304 DATA "St Emilion","3.48"
305 DATA "Rioja","3.50"
306 DATA "Vinho Verde","2.88"
307 DATA "Beaujolais Blanc","3.
    83"
308 DATA "Volnay","7.48"
309 DATA "Cotes du Rhone","3.07
    "
310 DATA "Rose d'Anjou","3.07"
311 DATA "Sauternes","4.42"
312 DATA "Monbazillac","3.40"
313 DATA "Gewurtz Taminer","5.1
    7"
314 DATA "Wehlener Sonnenuhr","
    3.92"
```

Fig. 3.6.

en la lista. Si × no tiene ninguno de estos valores, se producirá un error OUT OF RANGE. Esta característica suele poder utilizarse también para GOSUB$, pero en Sinclair BASIC es necesario calcular el número de línea utilizado como un objetivo. Con frecuencia, ello se consigue utilizando las funciones booleanas como se muestra en el último capítulo, pero, en otras ocasiones, los valores de una variable pueden utilizarse como un índice directo, sobre todo si esta característica se tiene presente cuando se planifican los números de línea en los que se colocarán subsecciones importantes de nuestro programa. Muchos de los programas contenidos en este libro son controlados por menú, lo que significa que una lista de opciones se presentará como en el programa de lista de vinos, pero la respuesta del usuario se utilizará también para dirigir el control del programa a una de una tabla de subrutinas. Colocando estas rutinas en las líneas 1000,2000,3000, y así sucesivamente, podremos tomar la elección del usuario y efectuar la sentencia CASE (ver figura 3.7) con:

```
200 GOSUB 1000*ch
```

siempre que no tengamos más de nueve opciones (si las tuviéramos, simplemente tendremos que empaquetarles más estrechamente).

3.6 Búsqueda

El almacenamiento de cantidades bastante grandes de datos significará que necesitamos medios de programación para encontrar un registro individual que se adapte a nuestras exigencias. Recordaremos el nombre y desearemos buscar el número de teléfono, por lo que tendremos que obtener nuestro programa para examinar la lista y encontrarle por nosotros. La forma más fácil (pero no la más rápida) utiliza una búsqueda en serie, comenzando al principio del fichero y examinando sucesivamente cada registro hasta encontrar una "coincidencia". Supongamos que deseamos examinar una matriz de nombres para encontrar el de "Julia" entre ellos. Probablemente, podríamos planificar el ejercicio como un pequeño bucle REPEAT:

```
PROC Encontrar la señora
//Los nombres están almacenados en una matriz n$(5,10)
//Tenemos que examinar los seis primeros caracteres de cada ca-
   dena en la matriz//
LET conteo = 0
REPEAT
   LET conteo = conteo + 1
```

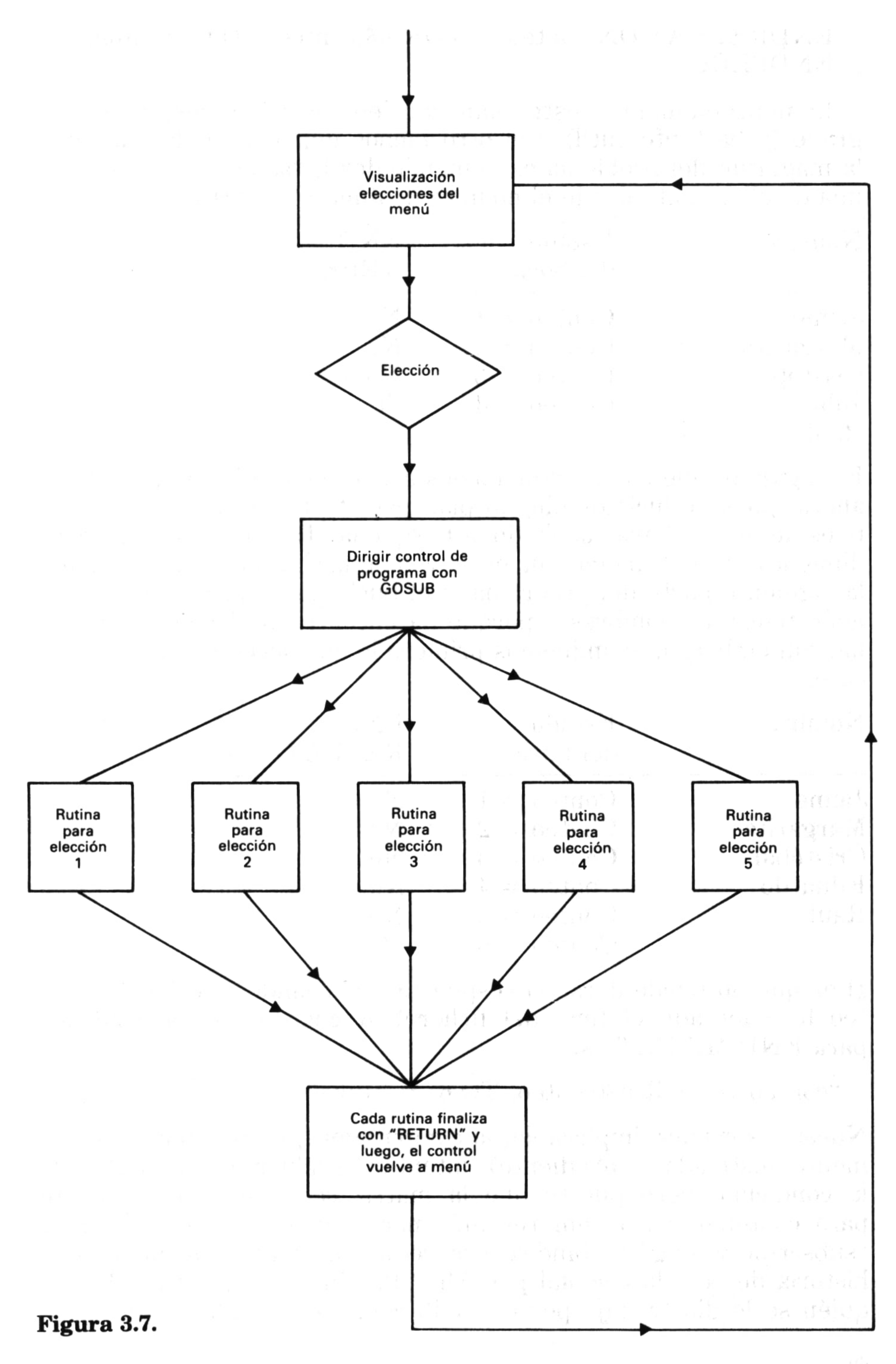

Visualización elecciones del menú
Elección
Dirigir control de programa con GOSUB
Rutina para elección 1
Rutina para elección 2
Rutina para elección 3
Rutina para elección 4
Rutina para elección 5
Cada rutina finaliza con "RETURN" y luego, el control vuelve a menú

Figura 3.7.

```
ENDREPEAT ON conteo > 5 OR n$(conteo, TO 6) = "Julia"
ENDPROC
```

Lamentablemente, este plan contiene dos errores, uno muy grave (y bastante sutil) y el otro menos importante. Reduzcamos la magnitud del problema examinando dos listas de cinco nombres, una de ellas conteniendo el nombre de Julia y la otra no.

Nombre	Estado del bucle	END REPEAT?
Jaime	Conteo = 1	No
Margarita	Conteo = 2	No
Cristóbal	Conteo = 3	No
Julia	Conteo = 4	Sí
Raúl		

El segundo, que es un error menos grave, podría hacerse evidente ahora, pues no hicimos ningún plan para tratar lo que sucede después de que se haya concluido la búsqueda. Tal como está el procedimiento en este momento, no hay ninguna prueba que indique a la siguiente parte del programa si el bucle ha finalizado porque ha leído todos los nombres o porque ha encontrado el nombre de Julia. Sin embargo, examinemos primero lo que sucede en el segundo caso.

Nombre	Estado del bucle	END REPEAT?
Jaime	Conteo = 1	No
Margarita	Conteo = 2	No
Cristóbal	Conteo = 3	No
Eduardo	Conteo = 4	No
Raúl	Conteo = 5	No
	Conteo = 6	??

¿Por qué no puede darse la respuesta "sí" cuando el valor del conteo ha superado el final del fichero? Recuerde que la condición para ENDREPEAT es:

"conteo > 5 OR n$(conteo, TO 6) = "Julia"

Nuestro sirviente implacable, la computadora, en una forma típicamente obstinada (y obediente), intentará evaluar ambas partes de la condición, pero puesto que la matriz se estableció solamente para contener cinco nombres informará, en este punto, un error "subscript wrong" (subíndice erróneo). Ello trae a la mente una historia de la vida colonial por Alice Perrin en la que Ram Din, a quién se le dio trabajo por el Sahib como un lavaplatos servil, fi-

nalmente "devuelve la pelota" a su brutal amo. Recibió instrucciones para encerrar a un sirviente inferior en un cuarto hasta que su amo regrese de lo que se esperaba fuera un corto viaje, Ram Din sigue al pie de la letra y deja al pobre criado que muera de inanición al producirse un retraso imprevisto, culpando de la muerte a las estrictas instrucciones de su amo.

Lo más grave es que los programas que implican ficheros, o arrays, requieren un cuidado particular cuando puede alcanzarse el final del fichero; una "ejecución en seco" tal como la intentada anteriormente suele poner al descubierto defectos en un plan aparentemente bueno.

```
PROC Encontrar la señora
//Este es correcto//
//Nombres en un array n$(5,10)//
LET conteo = 1
REPEAT
  IF n$(conteo, TO 6) = "Julia" THEN
    EXIT
  ENDIF
  LET conteo = conteo + 1
ENDREPEAT ON conteo > 5
ENDPROC
```

El programa (figura 3.8) que interpreta el pseudocódigo utiliza un bucle de una sola línea, probando primero el nombre y aplicando luego una prueba booleana con respecto al desbordamiento de la capacidad del fichero, lo que obliga a una salida directamente al mensaje de PRINT correspondiente. Constituye la base, con pequeñas variaciones, para varias rutinas de esta naturaleza contenidas en los programas de trabajo sucesivos a este libro.

3.7 Búsqueda y análisis de ficheros de texto

La mayor parte de los ficheros en este libro están en la forma de lista, pero los ficheros de texto sencillos se presentan con tanta frecuencia que se darán ahora unos pocos programas que utilizan técnicas de búsqueda. Por fichero de texto me estoy refiriendo a la clase de información encontrada en este libro (palabras separadas por espacios dispuestas en sentencias y párrafos, etc).

3.8 Búsqueda de cadenas

Este corto programa obtiene una cadena de texto y una cadena de búsqueda y luego, examina la cadena de texto para encontrar una

```
 10 Busquedas de nombres********
 20 GO SUB 200: REM Leer nombres
100 REM Bucle de busquedad en 120***
110 LET conteo=1
120 IF n$(conteo, TO 6)
    <>"Julia" THEN
     LET conteo=conteo+1:
      GO TO 120+20*(conteo>5)
130 PRINT "Encontrado en ";conteo:
    STOP
140 PRINT "No encontrado"
150 STOP
210 DIM n$(5,10): FOR i=1 TO 5
220 RETURN
300 DATA "Jaime"
301 DATA "Margarita"
302 DATA "Cristobal"
303 DATA "Eduardo"
304 DATA "Raul"
```

Figura 3.8.

```
 10 REM Busqueda en cadenas *******
 20 GO SUB 200: REM obtener cadenas
100 LET ls=LEN s$:
     LET tmax=lt-ls+1:
      LET tcnt=1
110 IF tcnt>tmax THEN
     PRINT "no hallada": STOP
120 IF t$(tcnt TO tcnt+ls-1)
    =s$ THEN GO TO 140
130 LET tcnt=tcnt+1: GO TO 110
140 PRINT "Hallada en ";tcnt:
    STOP
200 INPUT "texto";t$:
     INPUT "busqueda cadena";s$:
      RETURN

Listado de Variables
--------------------

t$     Cadena de texto a buscar
s$     Cadena a buscar
ls     Longitud de s$
lt     Longitud de t$
tmax   Ultimo caracter t$ comprobado
tcnt   Indice en t$
```

Figura 3.9.

coincidencia para la cadena de búsqueda; normalmente sería buscando una palabra en una sentencia o párrafo. Dirige una búsqueda en serie pero debe evitarse mirar más allá del último punto posible en el que puede encontrarse la cadena; por consiguiente, si la sentencia tiene, por ejemplo, 40 caracteres de longitud y está buscando "alegre", debe detener su búsqueda después de 36 comparaciones. No se hizo ninguna tentativa para comprobar si la coincidencia encontrada es una palabra completa, o solamente parte de una palabra, por lo que el programa informaría de un resultado satisfactorio cuando encuentra "canta" en "encantador". El listado del programa y las variables se muestran en la figura 3.9.

```
PROC Busqueda de cadenas
//Obtener t$ y s$//
//El bucle principal solamente se muestra aquí//
//Ver lista de variables para abreviaturas//
WHILE tcnt< = tmax AND t$(tcnt TO tcnt + scnt − 1) <> s$
  LET tcnt = tcnt + 1
ENDWHILE
```

3.9 Perfeccionamiento de la búsqueda de cadena

En la figura 3.10 se muestra un perfeccionamiento del programa anterior. En Mark2, tanto las cadenas de texto como las de búsqueda se convierten a letras minúsculas. Con ello se consigue, por ejemplo, que "la" coincida con "La". La parte principal del programa es prácticamente la misma, pero dos técnicas de programación de interés se encuentran en las subrutinas al final. La función de la última subrutina (línea 3ØØ) es convertir en minúsculas. Las letras mayúsculas tienen todas ellas CODIGOS entre 65 y 90 inclusive y en la cuarta sentencia de la línea 310, esto se prueba de forma booleana, dando lugar a un valor "uno" para la variable uc cuando se encuentra una letra mayúscula y "cero" en cualquier otro caso. Las letras minúsculas tienen CODIGOS mayores en 32 que sus contrapartidas mayúsculas y la siguiente sentencia utiliza esto para hacer la conversión cuando sea necesaria.

La rutina de conversión se utiliza tanto para las cadenas de texto como para las de búsqueda, que se realiza utilizando las variables i$ y o$. La única función de estas cadenas es transportar valores a, y desde, la rutina de conversión (i$ se pasa a la subrutina y o$ es objeto de retorno). Puesto que no se utilizan en ninguna otra parte en el programa se podrían llamar variables locales, una característica que algunos lenguajes de computadora ponen a disposición de cualquier variable, se utilice, o no, la misma etiqueta en otra parte del programa.

```
 10 REM Busqueda de cadenas MK2 *****
 20 GO SUB 200: REM Obtener cadenas
100 LET la=LEN a$
    LET lb=LEN b$:
    LET bmax=lb-la+1:
    LET bcnt=1
110 IF bcnt>bmax THEN
    PRINT "no hallada": STOP
120 IF b$(bcnt TO bcnt+la-1)
    =a$ THEN GO TO 140
130 LET bcnt=bcnt+1: GO TO 110
140 PRINT "Hallada en ";bcnt:
    STOP
200 INPUT "Cadena de texto";t$
    LET i$=t$: GO SUB 300
    LET b$=o$
210 INPUT "Cadena busqueda";s$
    LET i$=s$: GO SUB 300:
    LET a$=o$
220 RETURN
300 REM conversion a minusculas *****
310 LET o$="":
    FOR i=1 TO LEN i$:
    LET c=CODE i$(i):
    LET uc=((c-65)*(c-90)<=0):
    LET o$=o$+CHR$ (c+32*uc):
    NEXT i: RETURN
```

Figura 3.10.

3.10 Análisis de sentencias

Se trata de un programa sencillo (figura 3.11) que cuenta el número de palabras en una sentencia. Como se indica en la introducción, se enfrentará solamente con un formato bastante estricto de entrada: las palabras han de estar separadas por un solo espacio y la sentencia debe terminarse con un punto final. Para poder aliviar la anterior restricción sería necesario insertar una rutina de bucle en la línea 240 a la cual se pasa el control cuando el bucle principal encuentra un espacio o un punto final. La variable wc cuenta el número de palabras y cc el número de caracteres. Desde el punto de vista estructural, el programa está constituido por dos bucles, uno en el interior del otro. Dichos bucles suelen denominarse "anidados".

```
PROC Conteo de palabras
LET conteo_caracteres = 1
LET conteo_palabras = 0
```

```
REPEAT
  LET_conteo_palabras = conteo_palabras + 1
  REPEAT
    LET conteo_caracteres = conteo_caracteres + 1
  ENDREPEAT ON s$ (conteo_caracteres) = " " ORs$ (con-
teo_caracteres) = "."
  ENDREPEAT ON s$ (conteo_caracteres) = "."
  ENDPROC
```

```
 100 REM Analisis de sentencias*****
 110 CLS : PRINT : PRINT       "
Este programa analiza una
sentencia para determinar
cuantas palabras contien.
Cuando se visualice el
mensaje de solicitud,sirvase
introducir una sentencia.
Cerciorese de que exactamente
un solo espacio separa cada
palabra de la siguiente y
terminar su entrada con un
punto final."
 120 PRINT AT 20,3; FLASH 1;
    "Pulse cualquier tecla para
    continuar":
     PAUSE 0
130 PRINT AT 20,0;TAB 31;" "
140 PRINT AT 10,4; INVERSE 1;
    "Introduzca ahora su
    sentencia"
150 INPUT LINE s$
      PRINT AT 10,0;TAB 31
200 REM Bucles **************
210 LET cc=1: LET wc=0
220 LET wc=wc-1
230 LET cc=cc+1: IF s$(cc)<>" "
    AND s$(cc)<>"." THEN
     GO TO 230
240 IF s$(cc)<>"." THEN
     GO TO 220
400 PRINT                       "
El numero de palabras en su
sentencia es ";WC;"."
```

Figura 3.11.

3.11 Conteo de letras

El objetivo de este programa es obtener frecuencias para las ocurrencias de las letras del alfabeto en un fragmento de texto. Los resultados se almacenan en una matriz n() de dimensión 27, teniendo el último elemento el número de "otros caracteres" (espacios, signos de puntuación, etc.). Se hace uso de los CODIGOS asignados a caracteres para indexación en la n() matriz. Cada vez que se encuentra una, debe incrementarse el elemento de matriz correspondiente. El índice para la matriz n() se encuentra restando 64 (letras mayúsculas) o 96 (letras minúsculas) del valor del CODIGO. Si el carácter no es un miembro del alfabeto se utiliza el índice 27.

```
PROC Frecuencias de letras
//Obtener cadena de texto//
//La matriz n() contiene frecuencias//
WHILE conteo < = longitud_cadena
  LET asc = CODE 1$(conteo)
CASE de asc
  asc > 96 AND asc < 123
    //letra minúscula//
    LET índice = asc − 96
```

```
 10 DIM n(27): GO SUB 200
100 LET tot=0: LET indice=0:
    LET ln=LEN l$
110 FOR i=1 TO ln: LET indice=27
    :LET asc=CODE l$ (i)
120 IF asc>96 AND asc<123 THEN
     LET indice=asc-96
130 IF asc>64 AND asc<91 THEN
     LET indice=asc-64
140 LET n(indice)=n(indice)+1
150 NEXT i
160 FOR i=1 TO 26
     PRINT "letra ";
     CHR$ (i+96);" - ";n(i);
     "encontrada.":NEXT i
170 PRINT n(27);
    " otros caracteres."
180 PRINT : PRINT "Total ";ln;
    " caracteres."
190 STOP
200 INPUT "Cadenas texto:-";l$:
     PRINT l$: RETURN
```

Figura 3.12.

```
    asc > 64 AND asc < 91
      //letra mayúscula//
      LET índice = asc - 96
    OTHERWISE
      LET índice = 27
    ENCASE
    LET n(índice) = n(índice) + 1
  ENDWHILE
  ENDPROC
```

La rutina corta en el programa (figura 3.12, línea 16_) que imprime, de nuevo, los resultados hace uso de los valores del CO-

```
A lo lejos sobre las distantes
colinas los dorados rayos del
sol se habrian paso entre las
siniestras nubes. Los amigos
se dirigieron al albergue.

Letra a - 14 encontradas
Letra b -  4 encontradas
Letra c -  1 encontradas
Letra d -  4 encontradas
Letra e - 13 encontradas
Letra f -  0 encontradas
Letra g -  3 encontradas
Letra h -  0 encontradas
Letra i -  9 encontradas
Letra j -  1 encontradas
Letra k -  0 encontradas
Letra l - 11 encontradas
Letra m -  1 encontradas
Letra n -  7 encontradas
Letra o - 12 encontradas
Letra p -  1 encontradas
Letra q -  0 encontrodas
Letra r -  9 encontradas
Letra s - 20 encontradas
Letra t -  4 encontradas
Letra u -  2 encontradas
Letra v -  0 encontradas
Letra w -  0 encontradas
Letra x -  0 encontradas
Letra y -  1 encontradas
Letra z -  0 encontradas
21 otros caracteres

Total 128 caracteres
```

Figura 3.13.

DIGO, esta vez añadiendo 96 al valor de índice de la matriz. El número así obtenido se vuelve a convertir en la letra correspondiente por la función CHR$. La parte OTHERWISE de la construcción CASE se implanta preestableciendo el índice en 27 en la línea 11Ø y dejándolo invariable si no se encuentra ninguna letra del alfabeto en esa pasada del bucle. La salida impresa se muestra en la figura 3.13:

3.12 Sistema de consulta natural

El último programa en este capítulo es probablemente el más sofisticado de los contenidos en el libro. La información se mantiene sobre cinco individuos (el número podría haber sido mucho mayor); en el programa, ello se muestra en las sentencias DATA aunque se puede hacer en alguna otra forma.

De hecho, la primera acción tomada por el programa es leer (READ) todos los datos en una matriz bidimensional y en lo sucesivo, todo acceso tiene lugar a través de la matriz (array) 1$(). En acción, se le hace al usuario la pregunta 'A quién quiere conocer?' Entonces, el usuario puede introducir un apellido o parte del mismo. El programa trata de encontrar una coincidencia para cualquier cosa que se introduzca, bien sea "Martínez" bien sea simplemente "Martín" y si no se pudiera hacerlo así, responde con 'Can't find the name—try again' (no puede encontrar el nombre, pruebe de nuevo) hasta que lo consiga. Un procedimiento similar se sigue para el fragmento de información que quiera el usuario. Si esta información es la fecha de nacimiento, la expresión "fecha de nacimiento" puede introducirse completa o bien "fecha" o incluso "f". Siempre que pueda identificarse el requerimiento, se le dará al usuario la información completa.

La sofisticación radica en el encadenamiento por el cual se sigue la información, que es muy indirecto pero flexible. Da la computadora la apariencia no solamente de conocer lo que sabe sino también de estar enterada de sus áreas de ignorancia. Asimismo, parece tener una conjetura inteligente de las necesidades del usuario, al dársele una pequeña "pista". A partir de las entradas "F" y "t", la computadora responderá que "el número de teléfono de García es 55464"

La estructura de datos para el programa se muestra en la figura 3.14. Además de la matriz 1$(), que almacena los datos de objetivos, se utiliza otros dos. Las matrices a$() y b$() almacenan información sobre los campos en forma comprensible por la computadora y por el operador humano, respectivamente. Ambas tiene la dimensión principal 4, porque esa es el número de campos en la

Título campo—b$( )	apellido	nombre	fecha de nacimiento	número de teléfono
Número de campo—índice	1	2	3	4
Rebanada matriz—a$( )	l$ (r, 1 TO 12)	l$ (r, 13 TO 22)	l$ (r, 23 to 30)	l$ (r, 31 to 40)
1	Smith	Martínez Juan	12/09/53	76589
2	Jones	Ruiz José	08/12/56	77767
3	Fotheringay	García Alberto	03/03/54	55464
4	Little	Sánchez Luis	09/07/55	345269
5	Truebody	Rodríguez Raúl	08/02/60	23564

Figura 3.14.—Estructura de datos para sistema de consulta natural.

matriz objetivo. El índice proporciona un enlace entre a$() y b$(), por lo que cuando b$() es objeto de búsqueda para identificar el campo especificado por el usuario, el elemento correspondiente de a$() permite que la computadora evalúe el contenido de ese campo en la matriz objetivo 1$(). Por consiguiente, si el usuario introduce "ape", una búsqueda a través de b$() encuentra "apellido". El índice en b$() es 2 y aplicándole a a$() obtenemos "1$(r,13 TO 22)". Finalmente, VAL$ introduce la imagen y, dado un valor de r, evalúa

VAL$"$(r,13 TO 22)"

que es el apellido requerido.

La cadena de respuesta se muestra en forma de diagrama, en la figura 3.15. Se utilizó un método similar en PROFILE. Es muy potente y podría extenderse a situaciones en las que el usuario indica primero que tiene un número de teléfono, luego lo da y finalmente, extrae, por ejemplo, la fecha de nacimiento de la persona requerida. Solamente se hizo posible por los medios "en profundidad" del mecanismo de cálculo que el intérprete de Sinclair utiliza para las funciones de BASIC y, por supuesto, la aportación de la oscura, pero elegante, función VSL$.

Los bucles de búsqueda en el sistema de consulta ubicados en las líneas 120 a 150 y 170 a 190 (figura 3.16) respectivamente, siguen un plan económico pero no tan sencillo. Ambos utilizan la misma estructura por lo que la referencia se limitará al primero solamente. El mensaje que indica que se ha desbordado la capacidad del fichero (línea 120) y, por consiguiente, que el objetivo no

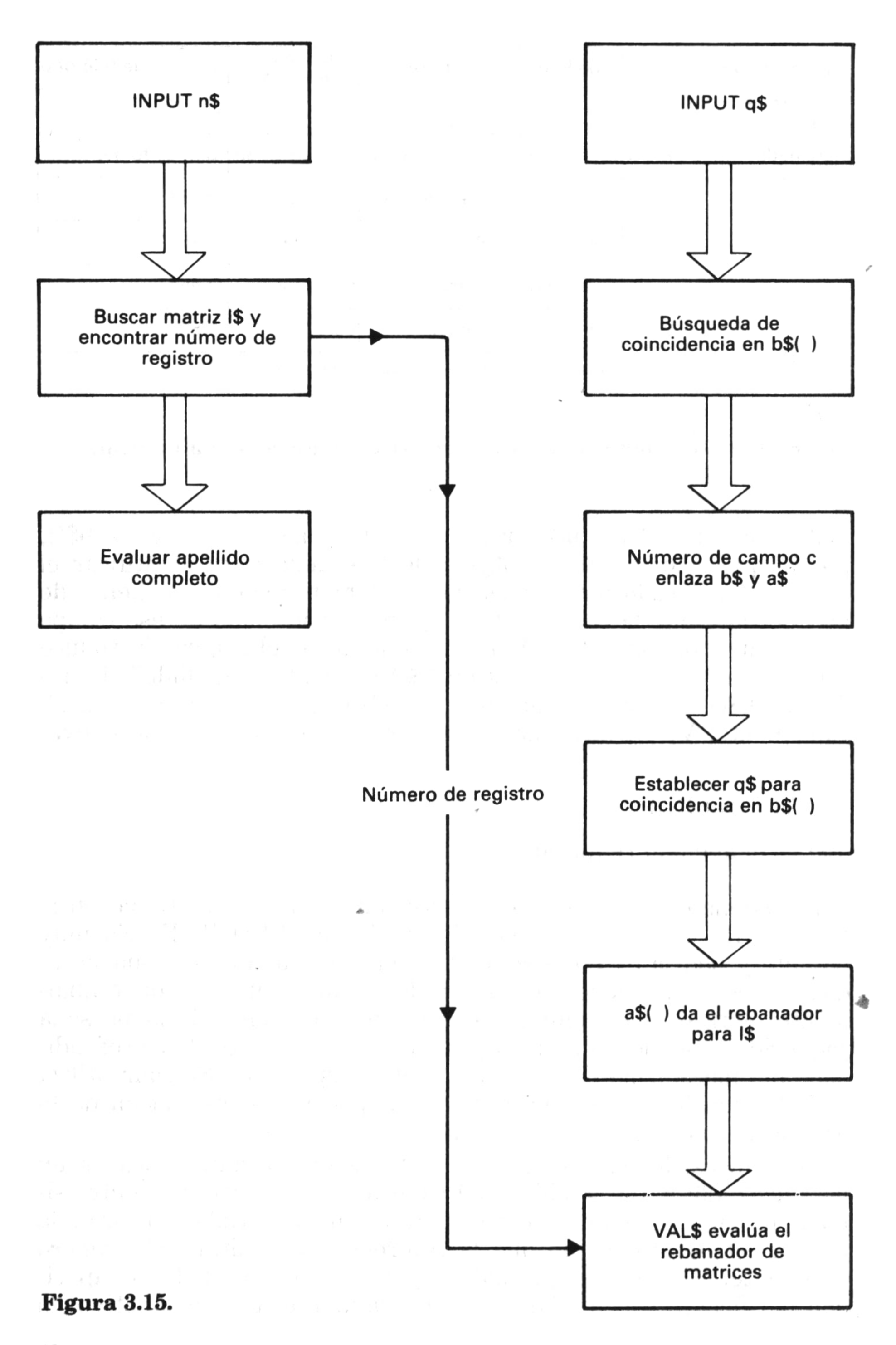

INPUT n$
Buscar matriz l$ y encontrar número de registro
Evaluar apellido completo
Número de registro
INPUT q$
Búsqueda de coincidencia en b$( )
Número de campo c enlaza b$ y a$
Establecer q$ para coincidencia en b$( )
a$( ) da el rebanador para l$
VAL$ evalúa el rebanador de matrices

Figura 3.15.

```
 10 DIM a$(4,11): DIM l$(5,40):
      DIM b$(4,16)
 20 LET b$(1)="apellidos":
      LET b$(2)="nombre"
 30 LET b$(3)="fecha nacim":
      LET B$(4)="numero telefono"
 40 LET a$(1)="l$(r,1 TO 12)"
 50 LET a$(2)="l$(r,13 TO 22)"
 60 LET a$(3)="l$(r,23 TO 30)"
 70 LET a$(4)="l$(r,31 TO 40)"
 80 GO SUB 300
100 REM Consulta natural *******
110 PRINT TAB 9; INVERSE 1;
     " Sistem consult": LET r=0
120 BEEP .2,0 : PRINT AT 20,0;
("No puedo encontrar el nombre
  pruebe de nuevo" AND (e=5)
130 PRINT AT 5,0;
"A quien quiere conocer?"
140 INPUT n$: LET ln=LEN n$:
      LET r=0: IF ln=0 THEN
       LET r=5: GO TO 130
150 IF r<5 THEN LET r=r+1:
       IF l$(r,1 TO ln)<>n$ THEN
        GO TO 150-30*(r=5)
160: PRINT AT 20,0;TAB 31;
      AT 7,0;
"Correcto. Que desea saber?"
      :let c=0
170 BEEP .2,0: PRINT AT 20,0;
("No se haga otro intento"
 AND(c=4))
180 INPUT q$: LET lq=LEN q$:
      LET c=0: IF lq=0 THEN
       LET c=5: GO TO 170
190 IF c<4 THEN LET c=c+1:
       IF b$(c,1 TO lq)<>q$ THEN
        GO TO 190-20*(C=4)
200 LET q$=b$(c)
210 PRINT AT 9,0;"La ";q$;
     " de ";VAL$ a$(1);" es";
     VAL$ a$(c)
220 STOP
300 REM Preparacion datos **********
310 FOR i=1 TO 5: READ
     l$(i,1 TO 12),
     l$(i,13 TO 22),
     l$(i,23 TO 30),
     l$(i,31 TO 40);
      NEXT i
320 RETURN
```

```
400 DATA "Martinez","Juan",
    "12/09/53","76589"
410 DATA "Ruiz","Jose",
    "08/12/56","77767"
420 DATA "Garcia","Alberto",
    "03/03/54","55464"
430 DATA "Sanchez","Luis",
    "09/07/55","345269"
440 DATA "Rodriguez","Raul",
    "08/02/60","23564"
```

Figura 3.16.

puede encontrarse, se coloca antes del bucle de búsqueda, pero se hace condicional en el valor del índice en la matriz. De este modo, la primera vez que se encuentra no es objeto de impresión, aunque si la búsqueda sale fuera del final del fichero (probado por la última sentencia en la línea del bucle 150) el control vuelve a la línea 120 con un valor de r que satisface ahora la condición. El efecto es que la rutina de búsqueda se repite hasta que se produzca una coincidencia. En programas sucesivos, utilizaremos búsquedas que puedan reanudarse después de la primera coincidencia para poder encontrar una segunda.

En este capítulo se han establecido la mayor parte de las técnicas básicas para la manipulación de datos. Ahora examinaremos los métodos de recogida de información, organización de esta última dentro de un fichero e introducción en un programa completo.

4 RECOGIDA, COMPROBACION Y ORGANIZACION

Hay muchas ocasiones en las que poco importa lo que escriba en una computadora. Si está jugando con el más reciente juego de invasores, puede sentirse herido en su orgullo personal si colisiona con un asteroide errante, pero esto no es nada en comparación con sus problemas si el programa de contabilidad personal muestra un gran crédito cuando, en realidad, está al borde de la quiebra. Grandes cantidades de tiempo y dinero se gastan por los profesionales de las computadoras en cerciorarse de que los hechos y las cifras que tienen en su haber son lo más exacto y fiable posible e incluso en el programa más modesto para su propio uso, hará lo máximo posible para dar alguna consideración a esta temática.

4.1 Entrada (INPUT)

En las pequeñas computadoras, prácticamente todas las entradas proceden del teclado. El primer factor de diseño debe ser hacer lo más claro posible lo que se está preguntando, con lo que disminuirá las posibilidades de respuestas sin sentido. El usuario debe ser capaz de ver lo que está tecleando y poder corregir los errores que pueda cometer. Cuando haya indicado el final de su entrada, normalmente la tecla ENTER, se le debe dar una posibilidad, si fuera posible, para comprobar y corregir nuevamente hasta que quede satisfecho. Finalmente, es de utilidad conseguir que la computadora compruebe la forma de entrada para ver si se adapta a la configuración correcta. Todo ello puede parecer bastante aburrido, pero hay algunos problemas fascinantes implicados y, en cualquier caso, este capítulo contiene algunas soluciones de aplicación inmediata que debe ser capaz de asimilar y utilizar en sus propios programas.

La versión de BASIC para el Spectrum, como la mayoría de los demás, proporciona una elección de sentencias de programa a utilizar durante esta fase: INPUT, INPUT LINE y INKEY$. No deseo duplicar el manual, pero puede ser de utilidad indicar en dónde cada una puede ser de mayor uso y para señalar sus desventajas.

INPUT permite que se dé un mensaje de solicitud y que aparezca en el fondo de la pantalla antes del cursor parpadeante, con

comillas si la variables es una cadena. La parte “inferior” de la pantalla es expansible por lo que, por ejemplo, si utilizamos:

```
100 INPUT AT 20,3;“Edad“;a$;AT 2,3;“Peso”;b$
```

la pantalla puede utilizarse con bastante flexibilidad. Un procesador de palabras modestísimo se obtiene por:

```
100 INPUT AT 20,0;“Pulse cualquier tecla y luego teclee lo anterior”;a$;AT 0,0;t$
```

Sin embargo, a medida que va descendiendo por la pantalla se hace más lenta la introducción por el teclado, lo mismo que los desplazamientos del cursor. Si teclea en demasía, se desplazará la parte superior de la pantalla, lo que es inconveniente para cualquier visualización que pueda haber puesto allí. Si hace una pulsación errónea de entrada correspondiente a una variable de número, el programa se detiene con un mensaje de error, lo que es desconcertante y si se borran las comillas suministradas para una variable de cadena, aparecerá un signo de interrogación parpadeante “?” hasta que corrija su error o teclee STOP. Todo ello puede prestarse a confusión y debe recordarse que al utilizar la computadora como una herramienta para almacenar información podemos estar atendiendo a un usuario no bien informado. La opción LINE con INPUT supera los problemas de las comillas y acepta toda información como variable de cadena, por lo que permite la conversión a tipo numérico por el programa cuando sea necesario. Sin embargo, no hay ninguna opción que permita la edición correctora de datos anteriormente introducidos, lo que, con respecto al almacenamiento de datos, es una desventaja.

INKEY$ aceptará solamente una pulsación de tecla cada vez y no espera a operadores vacilantes a no ser que lo combine con PAUSE 0, que hace que la computadora espere hasta que se pulse una tecla. Puesto que ello deja toda la tarea de comprobación y de impresión al programa, es la más flexible de todas las elecciones, siempre que sea correcta la segunda parte. El teclado de mi propio Spectrum ha tenido un trabajo notable en su corta vida y ahora descubro que:

```
PAUSE 0:LET a$ = INKEY$
```

no es tan fiable como:

```
PAUSE 0:LET c = PEEK 23560
```

que proporcionará el código (CODE) de la tecla pulsada. Ello ha de convertirse en una cadena para impresión pero tiene otras ventajas. Este es el método que utilizaré, con frecuencia, en el diseño de medios para la entrada.

Ahora dejemos a un lado los problemas internos de BASIC y consideremos la solución ideal. Pasamos a teclear, por ejemplo, algunos nombres y direcciones. Todos tendrán su propia imagen de cómo esto podría hacerse mejor. Daré una solución completa y parte de otra (solamente una parte, porque para ponerla en práctica a mi satisfacción requeriría un programa en código de máquina demasiado largo para poderlo incorporar en este libro. Una tercera posibilidad se da en el último programa en el capítulo 6.

El primer método es uno de eficacia probada en términos de cómputo (en realidad, con relación a la utilización general). Una forma se presenta en la pantalla, con adecuados, pero breves, mensajes de solicitud para cada área que requiera una respuesta, y el usuario es capaz de introducir por el teclado la información requerida. El tamaño de la pantalla (sobre todo su anchura) será un factor limitador, pues cada campo tendrá que encajar en una sola línea, pero podemos aceptarlo de momento. Un breve programa, procedente del manual, da la rutina de entrada básica:

```
10 PAUSE 0:PRINT INKEY$;:GOTO 10
```

Ello le permite teclear como casi con un máquina de escribir, incluso utilizar las teclas del cursor a la derecha y a izquierda, aunque llegue a ser algo aturdidor pues al cabo de algún tiempo no podrá decir en donde está en la pantalla. Se necesita un cursor y sería muy cómodo, si, a medida que se desplace a través de la línea de entrada, pudiera mostrar el carácter debajo de él. ENTER significará el final de la entrada en ese campo y nos desplazará a la siguiente. Todo lo que vayamos dejando detrás será registrado y al final de la forma, los campos constituirán juntos un registro que puede ser objeto de una llamada posterior para su visualización, edición o impresión. Para estas funciones (y la entrada, también) necesitaremos un menú similar al desarrollo en el capítulo 3. De forma abreviada, se trata de la especificación correspondiente al programa de la agenda de direcciones, que seguirá en partes.

4.2 Agenda de direcciones

En la figura 4.1 se muestra el menú. Cuando anteriormente, indexamos en la lista de elecciones (línea 220), pero esta vez, como un perfeccionamiento, utilizamos la sentencia RESTORE para imprimir la elección en la parte superior de la pantalla, con lo que se recordará al usuario lo que está haciendo. Pequeños retoques como éste aportará mucha diferencia para el usuario y ayudará a lograr una exactitud segura. Una subrutina, en la línea 900, confirma la elección, que puede considerar exigente. Hay que omitir la línea 260 y la

```
200 REM El menu ***************
210 CLS : LET y=1: LET n=0:
     PRINT 'TAB 9;"Agenda direcciones
"     'TAB 9;"___________________"
220 RESTORE 291: PRINT :
     FOR i=1 TO 4: READ m$:
      PRINT    TAB 4;i;" - ";m$
:      NEXT i
230 PRINT AT 20,2; FLASH 1;
   "Introduzca numero de su elec-
    cion"
240 PAUSE 0: LET a$=INKEY$:
     IF a$>"4" OR a$<"1" THEN
      BEEP .2,40: GO TO 240
250 LET ch=VAL a$: CLS :
     RESTORE 290+ch: READ m$:
      PRINT "Agenda dir-";
       INVERSE 1;m$;: BEEP .2,5
260 GO SUB 900:
     IF NOT ok THEN GO TO 200
280 PRINT AT 20.0;TAB 31:
     GO SUB ch*1000:
      IF ch<>4 THEN GO TO 200
290 STOP
291 DATA "Haga otra entrada"
292 DATA "Corrija una entrada"
293 DATA "Imprimir la lista"
294 DATA "Salida del programa"
900 REM Confirmacion *************
910 PRINT AT 20,2; FLASH 1;
" Confirme eleccion(y/n) "
920 PAUSE 0: LET ok=y:
     LET a$=INKEY$:
      IF a$<>"y" AND a$<>"Y"
       THEN LET ok=n
930 RETURN
```

```
         Agenda direcciones
         __________________

    1 - Haga otra entrada
    2 - Corrija una entrada
    3 - Imprimir la lista
    4 - Salida del programa


  Introduzca numero de su eleccion
```

Figura 4.1.

primera sentencia de 280 si lo hiciera. La variable OK(correcta) en la segunda parte de la línea 200 y la subrutina es un ejemplo de lo que se conoce como un indicador ("flag"-bandera). Encontraremos su utilización varias veces en este capítulo. Dicho indicador, o bandera, se eleva como una señal para mostrar que algo ha sucedido. Qué acción ha de tomarse depende de las circunstancias y, por lo general, la acción se retardará. En condiciones normales, el valor 1 significa que se ha producido el suceso y el valor 0 quiere decir que no ha tenido

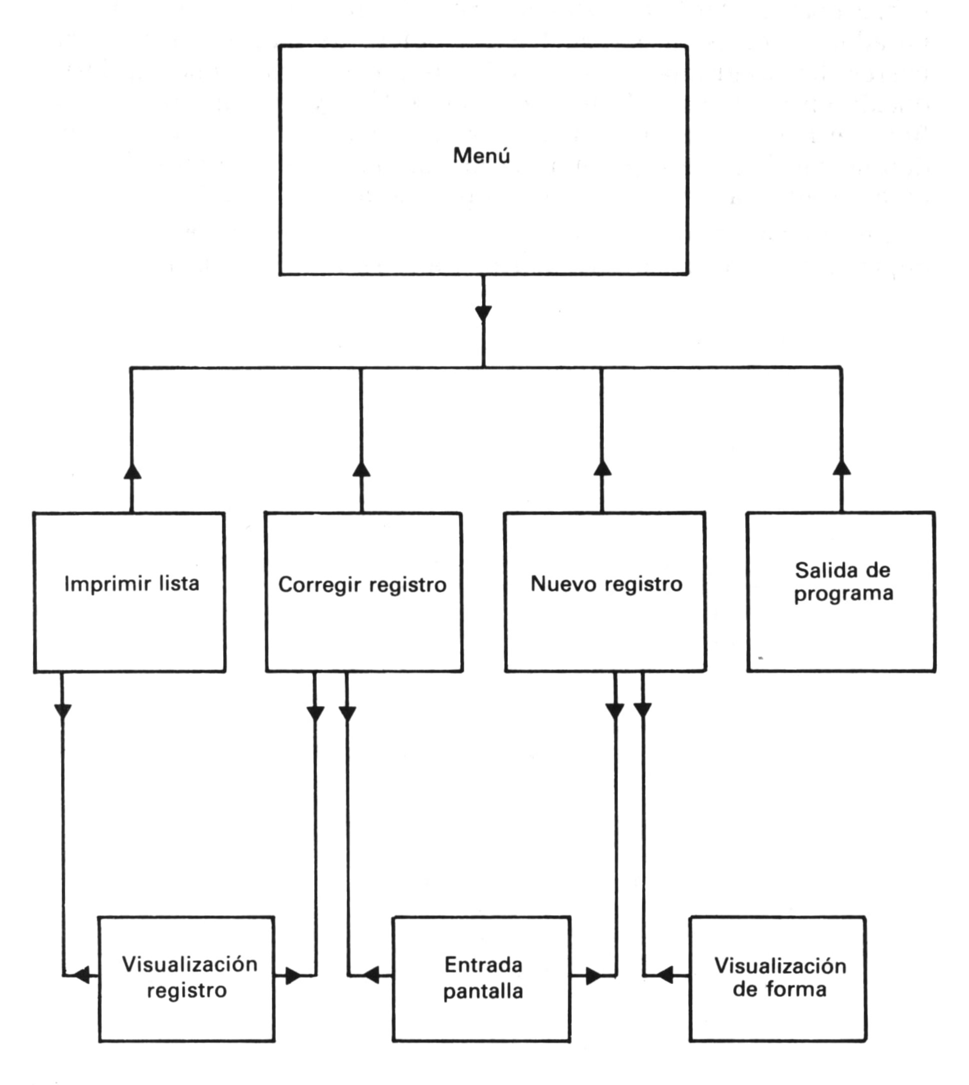


Figura 4.2.

lugar. En este caso, la "bandera" se eleva por la respuesta en la subrutina (la variable y se ha puesto a 1 en la línea 210) y nos permite volver a la lista de opciones si se hubiera producido un error. En la línea 260, indexamos en el propio programa (una tabla de subrutinas que llevarán a cabo las elecciones del menú). Si está tecleando estas rutinas, conserve (SAVE) la rutina del menú como propia ahora, pues se utilizará en otros programas en este libro y podría serle de gran utilidad para algunas de sus propias creaciones.

En la figura 4.2 se muestra el diseño total del programa. Las subrutinas de "utilidad" suelen tratar la interacción entre la computadora y el usuario y suelen ser objeto de llamada para varias partes del programa. Este tipo de diseño "modular" que fue introducido en el capítulo 1 tiene mucho de "ida y vuelta", pero es, al final, el más eficaz; muchos esfuerzos (y espacio de memoria) pueden desperdiciarse repitiendo la misma línea en diferentes lugares en el programa. ¡Además, soy un pésimo mecanógrafo!

Las subrutinas de edición y de nuevo registro requerirán que se imprima una forma, o formulario, como se indica en la figura 4.3.

```
400>REM Impresion de formulario *******
410 RESTORE 490: READ nf
420 FOR i=1 TO nf:
    READ p$,x,y,lf: DIM f$(lf)
430 PRINT AT x,y; INVERSE 1;p$;
    ">"; INVERSE 0;f$;
    INVERSE 1;"<": NEXT i:
    RETURN
450 REM Mostrar registro **************
460 RESTORE 490: READ nf:
    LET rr=0
470 FOR i=1 TO nf
    READ p$,x,y,lf: LET rl=rr+1
    : LET rr=rr+lf: DIM f$(lf):
    LET f$=l$(r,rl TO rr)
480 PRINT AT x,y; INVERSE 1;p$;
    ">"; INVERSE 0;f$
    INVERSE 1;"<":NEXT i:
    RETURN
490 DATA 7
491 DATA "Apellido",2,0,13
492 DATA "Es",2,22,4
493 DATA "Dir1",4,0,20
494 DATA "Dir2",6,0,20
495 DATA "Dir3",8,0,15
496 DATA "Dir4",10,0,12
497 DATA "Pcode",10,18,7
```

Figura 4.3.

Las dos subrutinas son muy similares y comparten el bloque de datos en las líneas 490 a 497, que es todo lo que ha de cambiarse para poder imprimir una forma diferente. Podría ser de utilidad para conservar (SAVE) esta parte particular del programa para su uso en otros. La primera sentencia DATA da el número de campos en la forma, la última da el mensaje de solicitud (p$), la fila y la columna en donde se imprimirá (x,y) y el número de caracteres asignados a los datos (1f). Si cambia los valores dados, recuerde que los valores x e y dan la posición para el comienzo del mensaje de solicitud. La colocación de dos campos en la misma línea requiere, pues, un poco de cálculo para asegurar que no estén en discrepancia. La variable (f$), utilizada para la recogida y edición de datos, es "reDIMensionada" al tamaño correcto en las líneas 420 y 470 (un proceso que no sería tolerado por muchas versiones de BASIC que permiten solamente una de dichas sentencias en un programa). En la primera subrutina se deja en blanco, en la segunda se extrae una "rebanada" del registro correspondiente utilizando las variables rl y rr. A medida que la rutina avanza a través

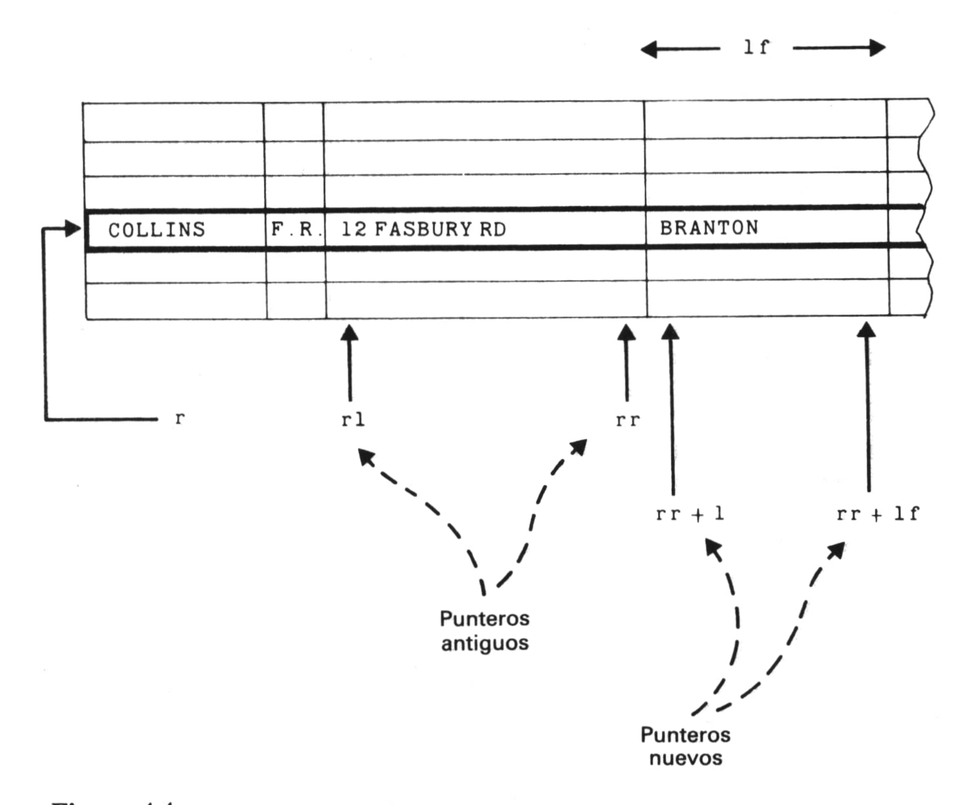


Figura 4.4.

de la forma, rl y rr se actualizan de modo que apunten automáticamente al elemento de datos que ha de ser objeto de visión (ver figura 4.4)

Los datos están almacenados en una matriz de dos dimensiones. La segunda dimensión es 91, pues este valor es el total de las longitudes de los campos y la primera es 50, un número bajo para fines de prueba. En un Spectrum de 48K, más de 300 registros podrían retenerse por este programa. La corta rutina que establece la matriz se muestra más adelante.

En la figura 4.5 (*página 57*) se muestra la forma y la rutina utilizada para entrada y edición. La parte principal se lleva a cabo por las líneas 120 a 170 y merecía la pena un estudio cuidadoso de su operación.

```
PROC Rellenar formulario
Visualizar el registro (en blanco si fuera nuevo)
Calcular límites derecho e izquierdo del campo
Poner posiciones antiguas y nuevas del cursor al principio del
   campo
REPEAT
   Ajustar nueva posición del cursor para que esté dentro del
      campo
   PRINT carácter en posición antigua del cursor
   PRINT cursor + carácter en nueva posición del cursor
   LET posición antigua del cursor = posición nueva del cursor
   PAUSE hasta que se pulse la tecla
   CASE
   La tecla es susceptible de impresión
      Hacer una rebanada en la variable de campo
      Añadir "uno" a la nueva posición del cursor
   La tecla se "borra"
      Hacer una rebanada de un espacio en la variable de campo
      Disminuir en "uno" la nueva posición del cursor
   La tecla no es ninguna de las anteriores y no "ENTER"
      BEEP (tono) de aviso
   ENDCASE
ENDREPEAT ON tecla es 'ENTER'
PRINT variable de campo
ENDPROC
```

Las variables utilizadas se muestran en la lista (figura 4.6. *Página 58*). La rutina se introduce en dos puntos diferentes. Cuando se introducen datos, la llamada en GOSUB 100, que asegura que la sentencia DIM en la línea 110 genere un campo en blanco, pero cuando es objeto de edición, se elude utilizando GOSUB 120, con lo que se permite el empleo del valor anterior de la variable de

```
100 REM Entrada pantalla *********
110 DIM f$(lf)
120 LET yl=y+LEN p$+1:
    PRINT AT x,yl;f$;:
    LET y=yl: LET oy=yl:
    LET yr=lf+yl-1
130 LET y=y+(yl-y)*(y<yl);
    LET y=y+(yr-y)*(y>yr);
    PRINT AT x,oy;f$(oy-yl+1);
   AT x,y; PAPER 4;f$(y-yl+1);
  : LET oy=y
140 PAUSE 0: LET c=PEEK 23560:
    LET a$=CHR$ c: BEEP .004,0:
    IF c>31 AND c<128 THEN
    LET f$(y-yl+1)=a$:
    LET y=y+1: GO TO 130
150 IF c=12 THEN
    LET f$(y=yl+1)=" ":
    LET y=y+1: GO TO 130
160 IF c=8 OR c=9 THEN
    LET y=y-(c=8)+(c=9):
    GO TO 130
170 IF c<>13 THEN BEEP.2,40:
    GO TO 130
180 PRINT AT x,yl;f$;: RETURN
```

```
Agenda de direcciones-Haga una nueva entrada

Apellido>Collins              <Es>E R <

Dir1 12 Fasbury Rd               <

Dir2 Branton                     <

Dir3 Middlewich              <

Dir4 Manchester  <Pcode>         <
```

Figura 4.5.

campo. La técnica de introducir subrutinas en puntos diferentes puede ser de gran utilidad pero ha de emplearse con reservas, pues, de no ser así, puede surgir complicaciones imprevistas.

Otro punto que requiere cuidado cuando se utilicen subrutinas múltiples es cerciorarse de que los nombres de variables empleados en una no están en discrepancia con los utilizados en otra. Cuando se obtuvo la lista de variables, descubrí, para mi desencanto, que había utilizado la variable y para dos fines diferentes en

```
Lista de variables
--------------------

nf              Numero de campos en registro
if              Contador
x               Linea pantalla
y               Col pantalla / Verdadero - 1
lf              Longitud de campo
yl              Limite izquierdo de campo
oy              Limite y antiguo
yr              Limite derecho de campo
c               Caracter tecleado
rl              Limite izquierdo en registro
rr              Limite derecho en registro
n               Falso - Ø
ch              Eleccion de menu
ok              Indicador (bandera)
cont            Indicador
max             Numero max de registros
rc              Conteo real de registros
r               Numero registro
l$( )           Matriz de registros
e$              Cadena edicion en campo
m$              Descripcion menu
r$              Cadena entrada en campo
p$              Solicitud campo
f$( )           Visualizacion en campo
a$              INKEY$
```

Figura 4.6.

este programa. Un examen detenido puso de manifiesto que realmente nunca estuvieron en discrepancia, pues nada fue cambiado y se tuvo la oportunidad de dar este aviso ¡puede ser que no tuviera tanta suerte!

Las subrutinas, que se llamaron del menú, se muestran ahora (figura 4.7, *página 59*). Ambas acceden al bloque de datos en las líneas 49Ø a 497, pero solamente para poder encontrar las longitudes de campos. La variable f$ se utiliza para "buscar y llevar", mientras que el registro se incorpora en r$ y e$ respectivamente hasta que, finalmente, los datos se colocan en la matriz maestra (l$). La rutina para modificación utiliza una forma bastante ardua e incómoda de encontrar el registro; una búsqueda adecuada habría sido mucho mejor. Programas sucesivos mostrarán a esos registros "en acción", así como métodos de clasificación que nos permitirán mantener registros en orden. En la figura 4.8 (*página 60*), la rutina de impresión se muestra con su visualización y dos secciones cortas, no incluidas en el menú, que se utilizan muy al prin-

```
1000 REM Nueva entrada *************
1010 IF rc=max THEN RETURN
1020 GO SUB 400: RESTORE 490:
     READ nf: LET r$=""
1030 FOR i=1 TO nf:
     READ p$,x,y,lf: GO SUB 100:
     LET r$=r$+f$: NEXT i
1040 LET rc=rc+1: LET l$(rc)=r$:
     RETURN
2000 REM Modificacion **************
2010 PRINT AT 20,1; FLASH 1; "
Que registro (1-50) a cambiar"
2020 INPUT r:
     IF r<1 OR r>rc THEN
     BEEP .1,40: GO TO 2020
2030 PRINT AT 20,0;TAB 31;:
     GO SUB 450
2040 RESTORE 490: READ nf:
     LET rr=0: LET e$=""
2050 FOR i=1 TO nf:
     READ p$,x,y,lf: LET rl=rr+1
    : LET rr=rr+lf: DIM f$(lf):
    LET f$=l$(r,rl TO rr)
2060 GO SUB 120: LET e$=e$+f$:
     NEXT i
2100 LET l$(r)=e$: RETURN
```

Figura 4.7.

cipio para establecer el tamaño de la matriz (cambio de la primera dimensión a un valor más grande) y cada vez que se salvaguarde (SAVE) el programa. La sección de impresión fue fácil de escribir puesto que todo el trabajo real se realiza por la subrutina en la línea 450, que visualiza un registro individual.

4.3 Un editor de pantalla completa

El editor, en el "corazón" del último programa, se le podría haber sugerido, como lo hizo conmigo, para la concepción de uno que permita el acceso a cualquier parte de la pantalla, utilizándole como una "ventana" en un fichero, registrándose todos los cambios, de forma automática, tal como se hicieron. La capacidad para "visionar" una colección de registros, en lugar de uno solo, también sería de utilidad. Esta sería una forma muy natural de examinar ficheros de datos; un bloque rectangular grande contendría toda la información, a razón de un registro por línea, con la posibilidad de

```
3000 REM Impresion lista ***********
3010 FOR r=1 TO rc: GO SUB 450
     PRINT AT 18,10:"Registro";
     r; TAB 31
3020 PRINT AT 20,2; FLASH 1;
     "Pulsar cualquier tecla para
      continuar": PAUSE 0:
     PRINT AT 20,0;TAB 31;:
     NEXT r
3030 RETURN
4000 REM Salida ********************
4010 SAVE "Agenda" LINE 200
4020 RETURN
5000 REM Comienzo lista ************
5010 DIM l$(50,91): LET max=50:
     LET rc=0: STOP
```

```
Agenda-Impresion de la lista

Apellido>Farmer          <Es>T   <

Dir1 143 Harwood Rd          <

Dir2 Southton                <

Dir3 Freeborough     <

Dir4 Hampsshire   <Pcode>T67 4AY<


        Registro 1

Pulse cualquier tecla para continuar
```

Figura 4.8.

que el usuario explore la superficie para examinar cualquier parte de los datos.

Esto fue lo que establecí, por mí mismo, cuando desarrollé PROFILE para el Spectrum de 48K. El programa fue objeto de prototipo en BASIC y luego, secciones grandes (incluyendo el editor, del que se da a continuación una versión a escala reducida) se convirtieron en rutinas de código de máquina para dar una respuesta instantánea.

El diseño del editor es muy similar al anteriormente dado para campos de la agenda de direcciones. En este caso, el cursor se desplazará en sentido horizontal y vertical a través de toda la pan-

talla y la "ventana de pantalla" se desplazará verticalmente, hacia arriba y abajo, el fichero, como a través de una escalera (en PROFILE se desplazará también en sentido horizontal). Los registros están restringidos a 32 caracteres por el segmento del programa mostrado en la figura 4.9 (*página 61*). Solamente es parte de una rutina más grande y para obtener la clase de visualización mostrada en la figura 4.10 (*página 62*) se necesita un pequeño trabajo de puesta a punto:

```
10 DIM l$(100,32) : LET x = 0 : LET y = 0
20 LET ox = 0 : LET oy = 0 : LET xo = 0 : LET oxo = 0
```

```
100 LET y=y*(y>=0)-(y=32):
    LET x=x+(x=0)-(x=21): PRINT
   AT 21,0; PAPER 2; INK 7;
  "Registro:";x+xO;TAB 31;" ";
110 PRINT AT ox,oy;
    l$(ox+xo,oy+1);AT x,y;
    PAPER 4;l$(x+xo,y+1):
    LET ox=x: LET oy=y
120 PAUSE 0: BEEP .004,-10:
    LET c=PEEK 23560:
    LET a$=CHR$ c:
    IF c>31 AND C<128 THEN
    LET l$(x+xo,y+1)=a$
    LET y=y+1: GO TO 100
140 IF c=12 THEN
    PRINT AT x,y;" ";:
    LET y=y-1:
    LET l$(xo+x,y+2)=" ":
    GO TO 100
150 IF (c-8)*(c-14)<=0 THEN
    LET Y=y*(c<>13)-(c-8)+(C=9):
    LETx=x-(c=11)+(c=10 OR c=13)
   : IF x>0 AND x<21 THEN
   GO TO 100
160 BEEP .2,40: LET oxo=xo:
    LET xo=xo+10*((x=21)-(x=0)):
    LET xo=xo-(xo-80)*(ox>=80):
    LET xq=xo AND (xo>0):
    IF oxo=xo THEN GO TO 100
170 BORDER 3: FOR i=1 TO 20:
    PRINT AT i,0;l$(xo+1):
    NEXT i: PAUSE 1; PRINT
   AT 0,0; PAPER 1; INK 7;  "
     Editor pantalla          "
  : GO TO 100
500 STOP
```

Figura 4.9.

```
          Editor Pantalla
Arroz descascarillado               1 Kg
Huevos                              36
Azucar                              4 Kg
Compota de fresa                    1 tarro
Pan en rebanadas                    2
Pan moreno pequeno                  1
Arroz blanco                        1 Kg
Bacon                               1 paquete
Nata (doble)                        .5 l
Mantequilla                         1 pequeno
Margarina                           .5 Kg
Sal                                 1 paquete
Detergente en polvo                 1 paq. grande
Detergente liquido                  1 paq. grande


  Registro:15
```

Figura 4.10.

Añada las líneas anteriores al programa y luego ejecútelo (RUN). Introduzca unos pocos caracteres, luego insértelos (BREAK) en el programa y suprima las líneas 10 y 20 (solamente se necesitan para preparar los valores iniciales). Teclee GOTO 180 para la reinicialización o relanzamiento. Cuando quiera desplazar la "ventana", mueva el cursor a la parte inferior de la pantalla con el empleo de las teclas de flechas y luego trate de seguir hacia abajo; la visualización se regenerará ("refrescará") como si la pantalla se hubiera deslizado hacia abajo en 10 segundos. Haga un desplazamiento hacia arriba de una forma semejante. Los movimientos del cursor y el accionamiento del teclado son algo lentos; para obtener la velocidad máxima, las líneas 100, 110 y 120 pueden combinarse todas ellas en una sola línea masiva 100. Como es habitual en estos casos, no ha de incluirse ni un solo espacio (que he utilizado simplemente para los fines de una disposición clara). Cuando quiera conservar (SAVE) el programa introdúzcale (BREAK) en la forma usual, y utilice SAVE "ventana" LINE 170. Como con todos los programas de almacenamiento de datos, nunca lo ejecute (RUN) después de que haya recogido los datos, a no ser que tenga la seguridad de que quiere borrar toda la información. Pueden añadirse medios suplementarios utilizando un menú en, por ejemplo, la línea 300; en el PROFILE las búsquedas, las sustituciones y las salidas impresas pueden realizarse, de forma automática, con el empleo de los nombres de campos asignados por el usuario.

4.4 Comprobación de datos

Uno de los mayores problemas en el mantenimiento de los ficheros es su salvaguardia contra datos que sean inadecuados. Poco puede hacerse mediante programas para tener protección contra datos que sean simplemente no verdaderos (así, por ejemplo, si me quito cinco años de mi edad no hay forma de que la computadora pueda saberlo), pero puede, y debe, hacerse mucho para retener una información que pudiera no ser cierta. Una fecha de nacimiento, tal como 29 de febrero de 1981, puede marcarse por medios mecánicos y visualizarse los mensajes correspondientes. Quizás más enojosa sea la introducción de números; por ejemplo, podemos desear realizar cálculos con datos numéricos y si se hubieran introducido de forma incorrecta (digamos, con números "0" en lugar de letras "ceros"), el programa fallará imprevistamente y quizás con resultados embarazosos. Todos los medios de entrada en este capítulo darían lugar a números que se almacenan como cadenas, por lo que para cálculos serían objeto de conversión con el empleo de la función VAL, que simplemente rehusa trabajar si encuentra SO en lugar de 50.

4.5 Validación de fechas (Valfech)

El proceso de comprobación mecánica para datos evidentemente inexactos se denominan validación. El siguiente programa (fig. 4.11) contiene comprobaciones para el número de días en un mes incluyendo los años venideros hasta el año 2000, en donde se ha observado el formato habitual de "día/mes/año". Está escrito de modo que pueda introducir una fecha para fines de comprobación y obtener una respuesta inmediata, pero podría volverse a numerar y utilizar como una subrutina. Se utiliza un indicador 'flag' (variable if) con el convenio de que:

ef = Ø significa que no hay error (la línea 1ØØ lo pone a este valor)
ef = 1 significa que se ha detectado un error

El indicador de bandera ('flag') se utiliza para controlar la salida impresa en la línea 19Ø; de cualquier otro modo, poca explicación se necesita.

4.6 Validación de números (Valnum)

La rutina para la validación de números es más larga y utiliza no menos de seis indicadores, cuyas combinaciones indican que se ha producido un error. Comprobará todos los números en el formato normal, con o sin lugares decimales, signos y ceros a la izquierda o a la derecha. Sin embargo, no tratará la notación "científica", pero

```
 10 DIM d$(8): DIM d(12):
    GO SUB 200
100 LET ef=0:
     IF d$(3)+d$(6)<>"//" THEN
     LET ef=1: GO TO 190
110 FOR i=1 TO 7 STEP 3:
     FOR j=0 TO 1:
   IF d$(i+j)<"0" OR d$(i+j)>"9"
   THEN LET ef=1: GO TO 190
120 NEXT j: NEXT i
130 LET m=VAL d$(4 TO 5):
     IF m=0 OR m>12 THEN
     LET ef=1: GO TO 190
140 LET y=VAL d$(7 TO ):
    LET d(2)=d(2)+(y=4*INT (y/4))
180 LET d=VAL d$( TO 2):
     IF d=0 OR d>d(m) THEN
     LET ef=1
190 PRINT ("No " AND NOT ef);
    "Error"
199 STOP
200 INPUT d$
210 FOR i=1 TO 12: READ d(i):
    NEXT i: RETURN
300 DATA 31,28,31,30,31,30,31,
   31,30,31,30,31
```

Figura 4.11.

retiene un error que Sinclair BASIC no lo hace (Pruebe 100 LET a = VAL("349.98") : PRINT a.). Los indicadores son:

digitflag	Se ha encontrado un dígito de cero a nueve
foundflag	Se encontró algo que no es espacio
dpflag	Punto decimal encontrado
signflag	+ 0 − encontrado (solamente se permite uno)
endflag	Espacio después del último carácter
errorflag	La cadena no es un número válido

Las reglas seguidas son:

1. No se permiten caracteres distintos de +, −,., 0−, y espacio
2. Al menos uno de 0−9 debe estar presente
3. Si se utiliza un signo (+ o −) debe ser al principio
4. No se permiten espacios entre otros caracteres
5. Solamente se permiten un punto decimal y un signo −

PROC Validacion de números
LET conteo_caract = 0 : LET digitflag = 0 : LET foundflag=0

```
LET dpflag = Ø: LET signflag = Ø: LET endflag = Ø
LET errorflag = Ø
REPEAT
    LET conteo_caract = conteo_caract + 1
    IF conteo_ caract > len THEN
      IF digitflag =Ø THEN LET erroflag = 1
      EXIT
  ENDIF
  LET char$ = $ (conteo_caract)
  CASE
  char$ =
    IF foundflag = 1 AND endflag = Ø THEN
      LET endflag = 1
    ENDIF
  char$ = "+ " OR char$ = "–"
    IF foundflag = Ø THEN
      LET foundflag = 1: LET signflag= 1
    ELSE
      LET errorflag= 1
    ENDIF
  char$ = "."
    IF dpflag = 1 OR endflag = THEN
      LET errorflag = 1
    ELSE
      LET dpflag = 1: LET signflag = 1
      LET foundflag = 1
    ENDIF
  char$ >= "Ø" AND char$ <= "9"
    IF endflag = Ø THEN
      LET digitflag = 1; LET signflag = 1
      LET foundflag = 1
    ELSE
      LET errorflag = 1
    ENDIF
  OTHERWISE
    LET errorflag = 1
    ENDCASE
  ENDREPEAT ON errorflag = 1
```

El listado (fig 4.12) sigue la descripción de pseudo-código más en espíritu que al pie de la letra. Exige que el número se introduzca como una variable de cadena de 10 caracteres de longitud. Si grandes tablas de datos han de comprobarse de esta forma, una rutina de código de máquina es mucho más rápida. La mostrada en la figura 4.13 se desarrolló para PROFILE a partir del programa

```
 10 DIM l$(10): GO SUB 200
100 LET ff=0: LET sf=0:
    LET df=0: LET lf=0:
    LET nf=0: LET ef=0: LET c=0
110 LET c=c+1: IF c>10 THEN
    LET ef=1-nf: GO TO 190
120 LET c$=l$(c)
130 IF c$=" " THEN
    LET lf=(ff=1): GO TO 180
140 IF c$="+" OR c$="-" THEN
    LET ef=ff: LET ff=1:
    LET sf=1: GO TO 180
150 IF c$="." THEN
    LET ef=(df OR lf):
    LET df=1: LET sf=1:
    LET ff=1: GO TO 180
160 IF c$>="0" AND c$<="9" THEN
    LET ef=lf: LET nf=1-lf:
    LET sf=1-lf:LET ff=1-lf:
    GO TO 180
170 LET ef=1
180 IF NOT ef THEN GO TO 110
190 PRINT l$;
  ("No " AND NOT ef)+"Error ";:
  IF ef THEN PRINT
      "EN CARACTER ";c
199 STOP
200 RETURN "Cadena a comprobar";l$
RETURN
```

Figura 4.12.

de BASIC. El código es objeto de la función POKE en la zona normalmente ocupada por gráficos definidos por el número en el extremo superior de la memoria y una vez realizado esto, podría conservarse utilizando:

SAVE "valnum"CODE 65368,123

y cargarse (LOAD) automáticamente por cualquier programa que necesite esta característica operativa. La función POKE en la línea 210 pasa la longitud de cadena a la rutina y debe hacerse cada vez que se utiliza la rutina (la misma posición se emplea para almacenar el resultado). Es objeto de llamada por la línea 100, cuya primera parte es necesaria para poder establecer la posición de la cadena en memoria. El número dejado en la posición 65368 es 255 si se detecta un error; de cualquier otro modo, una combinación de los indicadores.

```
  5 REM Validacion de numeros ******
 10 DIN m$(10): GO SUB 200
100 LET m$=m$:
    RANDOMIZE USR 65369:
PRINT ("No " AND PEEK 65368=255)
    ;"Error"
199 STOP
200 FOR i-1 TO 123: READ a:
    POKE (65367+i),a: NEXT i
210 INPUT m$: POKE 65368,10:
    RETURN
300 DATA 0,17,0,0,58,88,255,95,
   28,42
302 DATA 77,92,43,29,202,201,
   255,35,126,254
304 DATA 32,32,13,203,66,40,
   242,203,90,32
306 DATA 238,203,218,195,101,
   255,254,43,40,4
308 DATA 254,45,32,18,205,192,
   255,203,66,32
310 DATA 57,203,74,32,53,203,
   194,203,202,195
312 DATA 101,255,254,46,32,20,
   205,192,255,203
314 DATA 66,40,4,203,82,32,31,
   203,210,203
316 DATA 202,203,194,195,101,
   255,254,58,242,196
318 DATA 255,254,48,250,196,
   255,205,192,255,203
320 DATA 226,195,167,255,203,
   90,200,193,122,50
322 DATA 88,255,201,203,98,40,
   247,62,255,50
324 DATA 88,255,201
```

Figura 4.13.

Una vez que una cadena ha pasado las comprobaciones hechas por uno u otro de los programas antes dados, puede convertirse, con seguridad, en un número adecuado utilizando VAL. Con frecuencia, hay otras limitaciones a comprobarse (que el número es positivo, o un número entero, o que está comprendido entre dos valores establecidos) pero suelen ser bastante sencillas.

4.7 Almacenamiento de datos

El lector puede preguntarse por qué he llegado a tales longitudes con el último programa, pero en el tipo de aplicación que estamos

considerando, es casi siempre más cómodo introducir y almacenar números como cadenas, junto con otros datos, mejor que asignar una matriz por separado. Hay que hacer elecciones cuando se escoja el tipo de cadena y la siguiente sección detalla algunos de los problemas.

El intérprete de Sinclair BASIC utiliza dos clases de cadenas para almacenamiento:

1. Matrices de caracteres o cadenas. Estas han de ser DIMensionadas antes de que se utilicen, determinándose el tamaño por esa sentencia. Una matriz unidimensional de caracteres (por ejemplo, DIM 1$ (1000) se iniciará reteniendo 1000 caracteres de espacios, que pueden cambiarse más adelante por fragmentación en la matriz con información de utilidad:

   LET 1$ (100 a 110) ="Smith E.J."

   Las matrices pueden tener cualquier número de dimensiones, cada una sin ningún límite que no sea el espacio de memoria total disponible, una vez que hayan tomado su asignación el programa y los diversos ficheros del sistema. Los programas más grandes en este libro permiten, como mínimo, 30000 caracteres a utilizarse para almacenamiento de datos en una máquina de 48K, por lo que DIN 1$ (300,100), DIM (400,75)... podría emplearse para almacenar matrices de registros. No hay ningún medio en BASIC para permitir que éstos cambien de tamaño por lo que, al menos a primera vista, este método es algo inflexible.
2. Una cadena simple así llamada, que puede crecer pero no contraerse, podría utilizarse. En este caso, no se requiere ninguna sentencia de DIMensión, necesitándose solamente una sentencia LET simple para establecer el valor inicial

   LER 1$ = "Smith E.J."

   En lo sucesivo, un programa puede "fragmentar" este tipo de variable, cambiando o leyendo su contenido, u otra sentencia LED puede emplearse para concatenar más datos al final de la cadena, con lo que se le hace más larga:

   LET n$ = 1$ (TO5)
   LET 1$ = 1$ + "Brown P.K."

Fragmentando en datos previamente introducidos nos permitirá modificar registros, antes objeto de entrada y, naturalmente, necesitamos también, de vez en cuando, añadir más información al final. Podría parecer que la cadena simple es la respuesta ideal. En el Spectrum de 48K, el espacio dejado para las variable cuando se

```
200 REM El menu ****************
210 BORDER 3: CLS :
    LET y=1: LET n=0: PRINT
     'TAB 8;"Listin telefonico"
     'TAB 8;"_________________"
220 RESTORE 291: PRINT :
      FOR i=1 TO 7: READ m$:
       PRINT    TAB 4;i;" - ";m$
 :      NEXT i
230 PRINT AT 20,2; FLASH 1;
     "Introduzca numero elegido"
240 PAUSE 0: LET a$=INKEY$:
      IF a$>"7" OR a$<"1" THEN
       BEEP .2,40: GO TO 240
250 LET ch=VAL a$:
      RESTORE 290+ch: READ m$:
       GO SUB 700
260 GO SUB 600:
      IF NOT ok THEN GO TO 200
280 PRINT AT 20,0;TAB 31:
      GO SUB ch*1000: IF cont
       THEN GO TO 200
290 LET cont=y: STOP
291 DATA "Anadir nuevos registros"
292 DATA "Borrar un registro"
293 DATA "Corregir un registro"
294 DATA "Hallar un registro"
295 DATA "Imprimir la lista"
296 DATA "Comenzar nueva lista"
297 DATA "Salida del programa"
500 REM Pulsar cualquier tecla ******
510 PRINT AT 20,2; FLASH 1;
   "Pulsar cualquier tecla para
    continuar":
    PAUSE 0:
     PRINT AT 20,0;TAB 31;:
      RETURN
600 REM Confirmacion ************
610 PRINT AT 20,2; FLASH 1;
 "Confirme entrada (y/n) ":
    BEEP .2,5
620 PAUSE 0: LET ok=y:
      LET a$=INKEY$:
       IF a$<>"y" AND a$<>"Y"
        THEN LET ok=n
630 PRINT AT 20,0;TAB 31;:
     RETURN
700 REM Encabezamiento ***********
710 CLS : PRINT              "
Listin telefono-"; INVERSE 1;m$:
       BEEP .2,5: RETURN
```

Figura 4.14.

haya introducido un programa moderadamente grande puede tener más de 30.000 caracteres. Además, podríamos argüir, puesto que la cadena crece a medida que se añaden registros, que nunca emplearemos más memoria que la que sea necesaria y cuando los programas estén conservados en cinta, no tendremos que esperar mientras que los octetos no utilizados, pero asignados, son objeto de escritura y lectura en memoria.

Lamentablemente, nuestro razonamiento es falso. La adición al final de una cadena simple no es un proceso tan complicado como podría parecer. El espacio dejado, después de haberse introducido un programa, se utiliza para almacenamiento de datos y para lo que se conoce como "espacio de trabajo". El intérprete emplea el espacio de trabajo para obtener resultados intermedios en casi la misma manera que podríamos utilizar un trozo de papel para el borrado de una carta antes de pasarle definitivamente el papel para cartas. El problema es que el intérprete debe tener suficiente memoria disponible para ambas versiones durante el proceso de concatenación (copia la cadena completa en el espacio de trabajo, construye una nueva versión y luego la copia en el espacio para variables). Ello significa que aunque pudiéramos **retener** una cadena muy grande en memoria no siempre podemos añadirla.

Para ver lo que ello significa, introduzca el siguiente programa en la computadora:

```
10 LET 1$ = " "
20 LET 1$ = 1$ + "1234567890": GOTO 20
```

Este comienza con una cadena de longitud cero y añade "registros", cada uno con diez caracteres de longitud, hasta que finalmente obtengamos un mensaje OUT OF MEMORY. Si teclea:

```
PRINT LEN 1$
```

después de ejecutar el programa obtendrá un resultado decepcionalmente bajo (aproximadamente una tercera parte de los 40.000 caracteres teóricamente disponibles con un programa tan pequeño y una máquina de 48K). Puede determinar cuanta memoria se desperdicia tecleando:

```
DIM1$(40000)
```

y comprobando que, de hecho, puede alterar el contenido de una cadena tan grande mediante, por ejemplo:

```
LET 1$(39999) = "d"
```

Es por esto por lo que, en este libro, todos los programas principales utilizan matrices de caracteres de longitud fija y con una o

dos dimensiones. El problema de la asignación de espacios se resolverá utilizando las funciones LOAD y SAVE para hacer "crecer" a las matrices de dimensiones "fijas" como y cuando sea necesario. En el apéndice se dan instrucciones para realizarlo. En sólo un caso, en donde se necesite la eficacia máxima absoluta al utilizar el espacio de memoria, un programa de código de máquina se utilizará para superar el problema del espacio de trabajos, que, incluso con matrices de longitud fija, se plantea cuando los registros en medio de matrices grandes han de suprimirse para dejar espacio libre para los nuevos. No le preocupe si su primera lectura de esta sección ha dado lugar solamente a una comprensión confusa del problema. Tenga presente que si sigue con los métodos dados en este libro, podrá ser capaz de concentrarse en la importante actividad de retener y de trabajar con sus datos sin temor de un fallo repentino.

4.8 Lista telefónica (Tellist)

El programa final de este capítulo mantiene una lista de nombres y de números de teléfono. Se incluye para mostrar las técnicas para la supresión y búsqueda de registros y también para la mejora de las rutinas SAVE y de la inicialización. Aunque podría utilizar la técnica de la forma para la recogida de datos desarrollada anteriormente en este capítulo, se ahorrará espacio en este libro utilizando un método más sencillo. Cada registro en el fichero estará constituido por dos campos, de 20 y 12 caracteres de longitud respectivamente, por lo que un solo registro apenas cabe en una línea de pantalla. La sentencia de dimensionamiento correspondiente está en la línea 6040 (por supuesto, puede modificar el número y las dimensiones de los campos para sus propios fines).

En la figura 4.14 se muestran las subrutinas de utilidad y menú. Pequeños desarrollos del programa de la agenda de direcciones significan la inclusión de dos nuevas subrutinas que anteriormente surgieron en varios lugares, pero la extensión principal es la de elecciones de menús adicionales. En la figura 4.15 se muestra una mejora con respecto a las rutinas simples, pero funcionales, utilizadas antes. Estas rutinas se utilizarán, con pequeños cambios, en los programas sucesivos. La forma en que la rutina de salida utiliza la variable de indicador ('flag'), cont, significa que deba dar salida utilizando 's' para la parada pues, de no ser así, el programa no se verificará (recuérdese que las variables son conservadas con el programa).

La subrutina para introducir un nuevo registro (figura 4.16) visualiza entradas en la pantalla a medida que se realizan, las anula si no están confirmadas como correctas y le pregunta si quiere con-

```
6000 REM Comenzar nueva lista *****
6010 PRINT AT 3,11; INVERSE 1;
   "AVISO";AT 4,0; INVERSE 0'"
La ultima version de su programa
se almacena en cinta,pero si
continua con esta opcion se
perderan todas las adiciones y
cambios hechos desde la carga
del programa "
6020 PRINT                     "
Pulse 'y' solamente si esta
seguro de que no perdera una
informacion importante. Usando
la opcion 7 puede almacenar sus
datos, luego volver a esta
opcion y comenzar una nueva
lista."
6030 GO SUB 600
6040 IF ok THEN DIM l$(500,32):
      LET rc=0: DIM n$(20):
       DIM t$(12)
6050 LET cont=y
6090 RETURN
7000 REM Salida del programa ***
7010 PRINT AT 2,0;          "Sir
vase insertar la grabadora de
cinta e insertar casete,prepa
rada para conservar juntos el
programa y la informacion"
7020 GO SUB 500
7030 CLS :
     SAVE "LISTA" LINE 100
7040 PRINT AT 20,0;TAB 31
7050 GO SUB 700:PRINT'        "
Si se detuviese ahora seria
conveniente verificar(VERIFY)
su programa. Si se encuentra
un error de carga de cinta
puede volver al menu principal
tecleando GOTO 100 ."''TAB 7;   "
(NO TECLEE RUN)"
7060 PRINT AT 20,4;INVERSE 1;"
Pulse 's' para parar ";AT 21,4;"a
"y cualquier otra para seguir"
7070 PAUSE 0: LET cont=y:
      IF INKEY$="s" THEN
      LET cont=n
7080 PRINT AT 20,0;TAB 31
     AT 21,0;TAB 31;: RETURN
```

Figura 4.15.

```
1000 REM Hacer nueva entrada *******
1010 PRINT AT 2,0;"Nombre";
     TAB 20;"Numero";: LET ec=0
1020 PRINT AT 20,2; INVERSE 1;
    "Sirvase introducir detalles a
     continuacion"
1030 INPUT "Nombre.-";n$:
     INPUT "Numero.-";t$
1050 PRINT AT 3+ec,0;n$;t$:
     GO SUB 600
1060 IF NOT ok THEN
     PRINT AT ec+3,0;TAB 31:
     GO TO 1020
1070 LET ec=ec+1: LET rc=rc+1:
     LET l$(rc)=n$+t$
1080 PRINT AT 20,4; FLASH 1;
    "Otra entrada (y/m)"
1090 PAUSE 0: LET a$=INKEY$:
     IF a$="y" OR a$="Y" THEN
     LET ec=ec*(ec<16):
     GO TO 1020
1100 RETURN
2000 REM Borrado de una entrada *******
5000 REM Impresion de la lista ********
5010 LET rn=0
5020 GO SUB 700: FOR i=0 TO 16:
     LET rn=rn+1: IF rn<rc THEN
     PRINT AT i+2,0;l$(rn)
5030 IF rn>rc THEN
    PRINT AT i+2,0;TAB 31;" ";
5040 NEXT i: GO SUB 500
5050 IF rn<rc THEN GO TO 5020
5060 RETURN
```

Figura 4.16.

tinuar introduciendo otra. Ello es de utilidad cuando va siguiendo su camino a través de una lista, aunque sea un poco exigente. La entrada en la matriz se efectúa en la línea 1070 (la variable rc cuenta el número de registros). El proceso de visualización de la lista de registros es bastante simple (línea 5000). El número de registros visualizados es objeto de conteo, de modo que no se sobreimpriman las cabeceras y los pies de columnas (también para evitar la temida característica de "scroll?"). Se visualizan en lotes de 16.

La siguiente rutina a escribir era la de encontrar un registro, en la línea 4000 (figura 4.17). Es fundamental para el programa, como conjunto, puesto que para el borrado, o alteración, primero tenemos que ser capaces de encontrar el registro. Se tuvieron pre-

```
4000 REM Hallazgo de una entrada
4010 PRINT AT 20,1; INVERSE 1; "
Introduzca nombre a encontrar
abajo"
4020 INPUT "Nombre:;s$:LET
     rn=1:LET sl=LEN s$
4030 PRINT AT 20,0;TAN 31;
     AT 20,10; FLASH 1;
     "BUSQUEDA"
4040 LET encontrado=n: FOR i=rn
     TO rc: IF s$(i,TO sl)
     THEN LET rn=i: LET i=
     rc+1: LET encontrado=y
4050 NEXT i
4060 IF NOT encontrado THEN
     PRINT INVERSE 1; AT 8,3;
     "Ninguna coincidencia mas
encontrada"; : GO TO 4190
4070 PRINT AT 8,11;"ENCONTRADO";
     AT 20,0;TAB 31
4080 PRINT AT 11,0; INVERSE 1;
     l$(rn);
4090 PRINT AT 20,4; FLASH 1;
     "Continuar busqueda (y/n)?"
4100 PAUSE 0: IF INKEY$="y" OR
     INKEY$="Y" THEN LET rn=rn+
1    :GO TO 4040
4190 GO SUB 500: RETURN
```

Figura 4.17

sentes dos factores. En primer lugar, los registros se han almacenado en orden de entrada simple, por lo que para encontrar uno no hay ninguna alternativa a una búsqueda en serie. En segundo lugar, el usuario no tiene necesidad de conocer exactamente lo que está buscando. Se le solicita al usuario un nombre, pero, de hecho, cualquier número de caracteres (menor que 32) puede introducirse en s$ (la cadena de búsqueda). En la línea 4040, la comparación se realiza utilizando solamente el número de caracteres introducidos por el usuario. Para una mayor rapidez, se utiliza un bucle FOR...NEXT y si no da resultado satisfactorios el primer registro coincidente, se reanuda la búsqueda en el registro siguiente a la primera coincidencia. Esa es la finalidad de la variable rn. El bucle se deja forzando el valor del contador por encima del límite superior, pero, en la reanudación, el contador será objeto de reposición.

La subrutina para el borrado en la línea 2000 (figura 4.18) identifica primero, de forma positiva, el registro a borrar (la subrutina

```
2000 REM Suprimir una entrada ****
2010 GO SUB 4000: IF NOT encon
     trado
     THEN RETURN
2020 PRINT AT 20,4; FLASH 1;
    "  Pulsar 'y' para borrar";
   AT 21,4;
  "cualquier otra tecla para
   retornar";
2030 PAUSE 0:
     AT 21,0; TAB 31;
2040 IF INKEY$<>"y"
     AND INKEY$<>"Y" THEN RETURN
2050 FOR i=rn TO rc:
     LET l$(i)=l$(i+1): NEXT i
2060 LET rc=rc-1
2070 GO SUB 500: RETURN
```

Figura 4.18.

```
3000 REM Modificar una entrada ***
3010 GO SUB 4000:IF NOT encontra
     do
     THEN RETURN
3020 PRINT AT 20,2; INVERSE 1;
   "Introduzca cambio luego"
      :  INPUT "Nombre";n$
      : INPUT "Numero";t$
3030 PRINT AT 13,0; INVERSE 1;
     n$+t$: GO SUB 600:
     IF NOT ok THEN  GO TO 3010
3040 PRINT AT 20,4; FLASH 1;
     "Pulse 'y'para cambiar  ";
     AT 21,4;
     "cualquier otra tecla para re
      tornar"
3050 PAUSE 0:
      PRINT AT 20,0;TAB 31;
      AT 21,0;TAB 31;:
      IF INKEY$<>"y"
      AND INKEY$<>"Y" THEN RETURN
3060 LET l$(rn)=n$+t$
3070 PRINT AT 11,0; INVERSE 1;
      n$+t$;AT 13,0;
      INVERSE 0;TAB 31;" "
3080 GO SUB 500: RETURN
```

Figura 4.19.

de búsqueda es objeto de llamada) y luego, requiere que se vuelva a pulsar "y" antes de que se tome cualquier acción. Es conveniente incorporar tantas salvaguardas como sea posible contra la eliminación o alteración no deseada de los registros. Las líneas activas son 2050 y 2060. En primer lugar, el registro no deseado se recubre mediante escritura por los sucesivos, llevándoles, en sentido descendente, uno a uno en el bucle y luego, el conteo de registros se disminuye en "uno". De hecho, ello deja una copia del último registro después del final del fichero, pero no será utilizado y se recubrirá con escritura cuando se añada uno nuevo. La misma práctica de identificación positiva se sigue en la rutina de modificación en la línea 3000 (ver figura 4.19), tanto en el hallazgo de un registro como en la comprobación de la alteración (ambas operaciones objeto de visualización en la pantalla). El cambio se realiza por simple sustitución (línea 3060).

El inconveniente con el programa tal como está es que los registros se mantienen en orden aleatorio. A medida que crece la lista, el tiempo que lleva la búsqueda se hará cada vez más importante. El remedio obvio es implantar una rutina de clasificación y en el siguiente capítulo examinaremos varios procesos de clasificación. Asimismo, veremos que cuando se ha clasificado una lista, el proceso de búsqueda de un registro se puede hacer mucho más rápido. En el capítulo final, se pondrá de manifiesto las formas de mantener la lista en orden, en cualquier momento.

5 COMPARACIÓN Y CLASIFICACIÓN

Los ficheros de texto pueden ser de varios tipos diferentes. Una de sus características más importante es el método mediante el cual serán objeto de acceso los registros individuales. En el caso de entrada de texto a un procesador de palabras, con miras a finalizar como prosa continua, tal como en la mayor parte de este libro, los registros individuales pueden estar constituidos por párrafos, sentencias o simplemente, por líneas físicas de texto. En este caso, será necesario recuperar el texto en una secuencia determinada; normalmente, pero no siempre, el orden en el que se introdujo. Cualquier tentativa de clasificar el fichero en orden númerico, o en algún otro orden "lógico", daría lugar, de forma inexorable, a que se alterara el significado del texto, posiblemente perdido para siempre. Los ficheros de programas de computadora son un ejemplo importante de acceso secuencial, aunque no necesariamente una entrada secuencial. Cada registro en un programa BASIC es una línea numerada que contiene una o más sentencias. Los programadores pueden (y suelen hacerlo) introducir las sentencias sin seguir un orden numérico, pero cuando el fichero se procesa por un intérprete debe seguirse un orden estricto o los resultados podrían ser realmente muy extraños.

Los ficheros cuyos registros se suelen procesar en un orden determinado se conocen como ficheros secuenciales aunque, en términos estrictos, es el método de acceso lo que está etiquetando y no el propio fichero. En la práctica, a menudo la única limitación es el medio que se utiliza para almacenar el fichero (la lectura de un fichero almacenado en cinta magnética fuera de su orden natural llevaría un tiempo excesivo). Los ficheros, cuyos registros pueden ser objeto de acceso en prácticamente cualquier orden, se conocen como de acceso aleatorio y se suelen almacenar en disco o, si son bastante pequeños, en la memoria principal de la computadora. Aún cuando el orden de procesamiento puede que no sea estrictamente crítico para los ficheros de acceso aleatorio puede ser de utilidad almacenar los registros en algún orden natural, o lógico, para hacer la salida más fácilmente asimilable desde el punto de vida humano. Si ello se consigue desplazando físicamente registros en el interior del fichero, el proceso se denomina clasificación. Si se permite que los registros permanezcan en posición, pero un

fichero por separado está incorporado para determinar el orden de acceso, se dice que el fichero ha sido indexado. Hay muchos métodos de realizar los dos procesos, de indexación y de clasificación, pero, en este capítulo, sólo examinaremos los principios básicos.

La base para uno u otro proceso ha de ser un sistema de ordenación ordinariamente convenido y utilizamos dos, numérico y alfabético, para la exclusión virtual de cualquier otro método. Aunque podamos estar muy familiarizados con ambos sistemas, pocas sorpresas pueden surgir cuando se realicen por computadoras, por lo que vale la pena observar detenidamente lo que sucede realmente.

La clasificación alfabética en términos humanos es bastante sencilla. Aprenderemos el orden alfabético de las letras como lo hacen los niños. Las listas alfabéticas se introducen de forma muy gradual, con mayor frecuencia por ejemplo que por instrucción, por lo que finalizamos con una comprensión casi intuitiva del método mediante el cual se producen. Como en el caso de muchas asimilaciones, hay puntos que quedan poco destacados o incluso completamente ignorados. El sentido común nos permite superarlo, pero para las computadoras todo debe estar claramente definido. El alfabeto de la computadora utiliza unos 96 símbolos "imprimibles" y ha sido más o menos normalizado desde 1968. El sistema ASCII (American Standard Code for Information Interchange) se muestra en la figura 5.1. y ha obtenido una gran aceptación. Es paralelo con el juego de caracteres ECMA (European Computer Manufacturers Association) que es, esencialmente, una variante de ASCII. De vez en cuando, los fabricantes de computadoras (sobre todo de máquinas más pequeñas) se han apartado radicalmente de

CONTROLES	0	1	2	3	4	5	6 PRINT	7	8 ⇦	9 ⇨	10 ⇩	11 ⇧	12	13 NEW LINE	14	15
	16 INK	17 PAPER	18 FLASH	19 BRIGHT	20 INVERSE	21 OVER	22 AT	23 TAB	24	25	26	27	28	29	30	31
	32	33 '	34 "	35 #	36 $	37 %	38 &	39 '	40 (	41)	42 *	43 +	44 ,	45 –	46 .	47 /
	48 Ø	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;	60 <	61 =	62 <	63 ?
	64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 0
	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [	92 \	93]	94 ↑	95 _
	96 £	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 ©

Figura 5.1.—Implantación de códigos ASCII en el ZX Spectrum.

estos convenios, pero su utilidad es ahora tan grande que se está haciendo raro. Si las computadoras han de comunicarse entre sí y utilizan máquinas tales como impresoras, una configuración no normalizada pronto se hará engorrosa.

La base para el código es que 256 configuraciones pueden formarse a partir de los ocho dígitos binarios utilizados como la unidad de información normalizada en una computadora. El bit de orden alto suele dejarse libre (originalmente se utilizó sobre todo para fines de comprobación, pero muchos otros usos se han concebido por programadores y diseñadores ingeniosos). Ello reduce el total disponible a 128, que puede considerarse ventajosamente en cinco grupos:

- I (0–31) Códigos de control. Estos códigos no suelen imprimirse en pantalla o papel, pero se utilizan para controlar las acciones de las máquinas implicadas. Salvo en unos pocos casos, el acuerdo entre fabricantes en cuanto a su interpretación es algo más bien inexistente.
- II (32–47) Signos de puntuación, comenzando con el espacio.
- III (48–64) Los numerales 0 a 9 seguidos por algunos pocos signos de puntuación.
- IV (65–96) Las letras del alfabeto inglés en mayúsculas, de nuevo seguidas por algunos signos de puntuación.
- V (97–127) Letras minúsculas del alfabeto. Los símbolos al final de este grupo están bastante lejos de ser normalizados.

Será inmediatamente obvio que el alfabeto de la computadora es mucho más grande que el que aprendimos cuando éramos niños. No solamente cada letra aparece dos veces (mayúsculas y minúsculas), sino que los numerales, así como los símbolos matemáticos y de puntuación, tienen todos ellos asignado un lugar. Este es el motivo de algunas de las sorpresas antes citadas. Al compilar una lista, podríamos ignorar la diferencia entre ‘A’ y ‘a’, pero en la codificación ASCII los dos símbolos son bastante distintos y para sorpresa de muchos, la letra mayúscula precede a su minúscula correspondiente. En los lenguajes de computadoras, el término “precede” se suele sustituir por “es menor que”, por lo que la sentencia de que ‘A’ va antes de ‘a’ se expresa por ‘A’ < ‘a’.

5.1 Comparación

Ahora tenemos una base para comparar cadenas de caracteres que, a su vez, permitirán que se preparen listas clasificadas. Dos cadenas, o secuencias de caracteres se comparan, carácter a carácter, hasta que se encuentre una diferencia. La posición relativa de

los caracteres encontrados en el código ASCII determinará el orden en que se colocarán las cadenas completas en la lista clasificada.

```
A r r e g l o
    ∧
A r t í s t i c o
```

La comparación anterior no encuentra diferencia alguna hasta que se llega a la tercera letra. Puesto que 'r' < 't', 'Arreglo' precederá a 'Artístico' en una lista clasificada. Sin embargo, obsérvese que "Artístico" irá antes de 'arreglo' en la misma lista, debido a la diferencia entre las versiones en mayúsculas y en minúsculas de la misma letra. En el caso de que una cadena sea más corta pero, por lo demás, idéntica a otra, como en :

```
M o d a
    ∧
M o d a l i d a d
```

podemos considerar la cadena más corta como rellenada con espacios a la derecha y por ello la comparación muestra una diferencia en el punto indicado. Puesto que un carácter de espacio en ASCII tiene el código más bajo de todos los símbolos susceptibles de impresión, la palabra más corta en estas circunstancias irá siempre primero.

Este proceso se ilustra a continuación en pseudocódigo. Las cadenas a$ y b$ son objeto de comparación y la más pequeña de las dos se almacena en una nueva variable f$. LEN y MIN son funciones previamente definidas que las daremos por sentado.

```
PROCEDIMIENTO Comparación
LET la = LEN(a$) : LET 1b = LEN(b$)
LET minlen = MIN(la,1b)
LET conteo _ caract = 1
WHILE conteo _ caract< =minlen AND
a$ (conteo _ caract) = b$ (conteo _ caract)
  LET conteo _caract = conteo _carct + 1
ENDWHILE
IF conteo _ carct = minlen + 1
  IF minlen = la
    LET f$ = a$
  ELSE
    LET f$ = b$
  ENDIF
ELSE
  IF a$(i) < b$(i)
```

```
    LET f$ = a$
  ELSE
    LET f$ = b$
  ENDIF
ENDIF
ENDPROC
```

En todos, con la excepción de los más pequeños intérpretes del BASIC, la primera parte de este procedimiento (la comparación de las dos cadenas) está "incorporada" (con elementos físicos), por lo que no hay necesidad de que se sirva dando un programa completo al nivel anterior. Muchas versiones del lenguaje permitirán que se programe la parte final como:

```
100 IF a$ < = b$ THEN LET f$ = a$ ELSE LET f$ = b$
```

Si la clásula ELSE no está disponible puede escribir en una sola línea utilizando:

```
100 LET f$ = a$ : IF a$ > b$ THEN LET f$ = b$
```

La función lógica en Sinclair BASIC puede emplearse para dar una versión compacta sin clásulas IF, por ejemplo:

```
100 LET f$ = a$ AND (a$ < = b$) + b$ AND (a$ > b$)
```

La ventaja de la versión anterior es que puede ir seguida por otras sentencias en la misma línea, pero obsérvese que la comparación ha de hacerse dos veces para obtener el valor de f$. Como se señaló en el capítulo 3, sentencias múltiples en una sola línea puede ser de utilización engorrosa cuando estén implicadas cláusulas IF. Si se están comparando números, podemos utilizar:

```
100 LET max = a + (b - a)*(b > a)
```

5.2 Clasificación

La sección anterior ilustra cómo se comparan dos cadenas y constituye la base de la mayoría de los procedimientos de clasificación, por la sencilla razón de que las computadoras trabajan completamente en un sistema binario y tienen que clasificar una lista por comparaciones continuadas de parejas. Mientras que el intelecto humano puede explorar unas cinco o seis palabras y colocarlas en orden en lo que al menos parezca ser un sólo proceso, la computadora está estrictamente limitada a las comparaciones por parejas.

Supongamos que hemos creado un fichero constituido por registros de longitud fija, que es suficientemente pequeño para la colección completa a retener en memoria. Para mayor sencillez, supon-

gamos que cada registro sólo tenga dos campos: APELLIDO al que asignaremos 15 caracteres y PROFESION con 20 caracteres. Nada se perderá en términos de principios generales si trabajamos con un número pequeño de registros, por ejemplo ocho.

Ficheros Nombres y profesiones

Apellido	Profesión
Sánchez	Carpintero
Ruiz	Carpintero
Ribera	Fontanero
Herrera	Electricista
Bernal	Fontanero
Zorrilla	Carpintero
Cruz	Electricista
García	Carpintero

Este fichero podría clasificarse de varias formas. Podríamos tomar el registro completo como la base para la comparación o selección de uno u otro campo, ignorando el contenido del otro. Cuando solamente se utiliza un campo como el criterio, se conoce como clave. El mismo término se emplea también si el factor decisorio es una combinación de caracteres procedentes de diversos campos; por ejemplo, los cinco primeros caracteres del apellido seguidos por los tres primeros de la profesión. Veremos, más adelante, cómo una elección cuidadosa de la clave puede llevar a resultados de utilidad, pero consideremos, de momento, un proceso que utiliza el registro completo para decidir una disposición final. El proceso no ha de ser como el que se utilizaría con ficheros grandes por razones evidentes, sino uno sencillo de comprender y que se parezca, lo más posible, a la forma en que podríamos, por nosotros mismos, trabajar con una tarea similar.

Comenzamos examinando el fichero completo para identificar el registro que irá primero en nuestra lista clasificada y luego, llevarle a la cabecera del fichero permutándole con el ya existente allí. Ahora podemos limitar nuestra atención a los siete nombres restantes y repetir el proceso, esta vez llevando nuestro registro seleccionado a la segunda posición. Es evidente que después de siete repeticiones la lista estará completamente clasificada (debiendo estar el último registro en su posición correcta si lo están todos los demás). En cada etapa, los registros tendrán nombres de variables 1$ (1). 1$ (2),...,1$ (8), pero el contenido de cada cadena cambiará a medida que avancemos en el proceso. Utilizaremos f$ como almacenamiento temporal para el registro de cadena esti-

mado primero en la lista en cualquier momento y rn como su posición en el fichero. Un recorrido a través del fichero para encontrar el siguiente registro a desplazar, en sentido ascendente, se denominará una pasada. Los resultados de las dos primeras pasadas se muestran a continuación.

Comienzo

Sánchez	Carpintero
Ruiz	Carpintero
Ribera	Fontanero
Herrera	Electricista
Bernal	Fontanero
Zorrilla	Carpintero
Cruz	Electricista
García	Carpintero

Después de la primera pasada

García	Carpintero
Ruiz	Carpintero
Ribera	Fontanero
Herrera	Electricista
Bernal	Fontanero
Zorrilla	Carpintero
Cruz	Electricista
García	Carpintero

García se identificó como el primer registro y se intercambió con la cabecera de fichero anterior (Sánchez).

Después de la segunda pasada

García	Carpintero
Bernal	Fontanero
Ribera	Fontanero
Herrera	Electricista
Ruiz	Carpintero
Zorrilla	Carpintero
Cruz	Electricista
Sánchez	Carpintero

Bernal y Ruiz se han intercambiado. En la siguiente pasada se desplazarán Ribera y Cruz.

```
PROC Clasificación de sustitución
LET pasada = 1
WHILE pasada < 8
  LET f$ = 1$(pasada)
  LET mas _ bajo _ hasta _ ahora = pasada
  LET siguiente _a _ comparar _ = pasada + 1
  // Examen del resto de la lista para encontrar el más bajo//
  WHILE siguiente _ a _ comparar < = 8
  IF f$ = 1$(siguiente _ a _ comparar)
      LET f$ = 1$(siguiente _ a _ comparar)
      LET mas _ bajo _ hasta _ ahora = siguiente _ a _ compa-
rar
    ENDIF
    LET siguiente _ a _ comparar = siguiente _ a _ comparar+1
  ENDWHILE
  // Ahora llevar el siguiente arriba permitiendo los registros//
  LET 1$(más _ bajo _ hasta _ ahora) = 1$(pasada)
  LET 1$(pasada) = f$
  LET pasada = pasada + 1
ENDWHILE
ENDPROC
```

```
 10 LET n1=8: GO SUB 230
100 REM Clasificacion de susti
     tucion
110 FOR p-1 TO n1-:
     LET f$=1$(p): LET 1sf=p:
      FOR c=p+1 TO n1
120 IF 1$(1sf)>1$(c) THEN
    LET 1sf=c
130 NEXT c: LET f$=1$(1sf):
     LET 1$(1sf)=1$(p)
      LET 1$(p)=f$: NEXT p
210 FOR i=1 TO n1: PRINT 1$(i):
     NEXT i
220 STOP
230 DIM 1$(n1,15)
240 FOR i=1 TO n1: READ 1$(i):
    NEXT i
250 RETURN
260 DATA "Sanchez"
270 DATA "Ruiz"
280 DATA "Ribera"
290 DATA "Herrera"
300 DATA "Bernal"
310 DATA "Zorrilla"
320 DATA "Cruz"
330 DATA "Garcia"
```

Figura 5.2.

El programa para una clasificación de sustitución se muestra en la figura 5.2. Una pequeña mejora puede hacerse retardando el intercambio de registros hasta que se haya examinado la lista completa. Ello se ilustra en la figura 5.3.

La rutina de clasificación de sustitución trabaja en una forma de sentido común, pero tiene la desventaja de que actúa a través de la lista completa, aún cuando ya pueda estar en orden. En la clasificación de burbuja (turbulencia) que se indica a continuación se toman medidas para permitir que la rutina termine tan pronto como la lista esté en orden, lo que puede dar lugar a un ahorro considerable, sobre todo si, como suele ocurrir, los elementos de datos se añaden a una lista ya clasificada

```
 10 LET nl=9: GO SUB 230
100 REM Sustitucion retardada
110 FOR p=1 TO nl-1
     LET f$=l$(p): LET lsf=p:
      FOR c=p+1 TO nl
120 IF l$(lsf) l$(c) THEN
     LET l$(lsf)=l$(p):
      LET l$(p)=f$: NEXT p
210 FOR i=1 TO nl: PRINT l$(i):
     NEXT i
220 STOP
230 DIM l$(nl,15)
240 FOR i=1 TO nl: READ l$(i):
     NEXT i
250 RETURN
260 DATA "Sanchez"
270 DATA "Ruiz"
280 DATA "Ribera"
290 DATA "Herrera"
300 DATA "Bernal"
310 DATA "Zorrilla"
320 DATA "Cruz"
330 DATA "Garcia"
340 DATA "Abajo"
```

Figura 5.3.

Esta clase de clasificación de burbuja lleva consigo más comparaciones continuadas de elementos de datos consecutivos en la lista. Siempre que se detecte que un par está en el orden equivocado, será objeto de intercambio. Si se realizan comparaciones desde la parte inferior de la lista hacia arriba, los elementos "más ligeros" subirán a la parte superior (de ahí su denominación de burbuja). Una clasificación que actúa desde arriba abajo se denomina una clasificación de sumidero.

Primera comparación

Sánchez	Carpintero	
Ruiz	Carpintero	
Ribera	Fontanero	
Herrera	Electricista	
Bernal	Fontanero	
Zorrilla	Carpintero	
Cruz	Electricista	← Cruz y García
García	Carpintero	← cambian lugares

Segunda comparación

Sánchez	Carpintero	
Ruiz	Carpintero	
Ribera	Fontanero	
Herrera	Electricista	
Bernal	Fontanero	
Zorrilla	Carpintero	←
García	Carpintero	←
Cruz	Electricista	

Tercera comparación

Sánchez	Carpintero	
Ruiz	Carpintero	
Ribera	Fontanero	
Herrera	Electricista	
Bernal	Fontanero	←
García	Carpintero	←
Zorrilla	Carpintero	
Cruz	Electricísta	

Cuarta comparación

Sánchez	Carpintero	
Ruiz	Carpintero	
Ribera	Fontanero	
Herrera	Electricista	←
García	Carpintero	←
Bernal	Fontanero	
Zorrilla	Carpintero	
Cruz	Electricista	

Podemos ver que, en este caso, García subirá, durante la primera "pasada", hasta que esté a la cabeza de la lista. En la segunda pasada, solamente se realizarán seis comparaciones llevando a Cruz a la segunda posición, pero no realizando una segunda comparación con García. Si, en cualquier pasada, no tienen lugar intercambios, el proceso se interrumpe bruscamente ("aborta"), puesto que el fichero debe estar en orden.

```
PROC Clasificación de burbuja
LET pasadas _ a _ hacer = 8
REPEAT
  LET permutaciones _ realizadas = Ø
  LET comparaciones _ a _ hacer = pasadas _ a _ hacer
  LET inferior _ de _ par = 8
  // comienzo en la parte inferior//
  WHILE COMPARACIONES _ a _ hacer > Ø
    LET superior _ de _ par = inferior _ de _ par − 1
    IF 1$(inferior _ de _ par) < 1$ (superior _ de _ par)
      //estan fuera de orden//
      LET f$ = 1$ (inferior _ de _ par)
      //por lo que les permutamos//
      LET 1$(inferior _ de _ par) = 1$(superior _ de _ par)
      LET 1$(superior _ de _ par) = f$
      LET permutaciones _ realizadas = permutaciones _ reali-
zadas + 1
      //y haremos una nota//
    ENDIF
    LET inferior _ de _ par = superior _ de _ par
    LET comparaciones_ de _hacer = comparaciones _ a _ ha-
cer − 1
ENDWHILE
LET pasadas _ a _hacer = pasadas _ a _ hacer − 1
IF permutaciones _ realizadas = Ø        //lista está en orden//
    LET pasadas _ a _ hacer = Ø          //por lo que se le da
salida//
   ENDIF
ENDREPEAT ON pasadas _ a _ hacer = Ø
ENDPROC
```

El programa que sigue este método casi "al pie de la letra" se muestra junto con sus resultados en la figura 5.4., pero, de nuevo, puede hacerse una mejora, esta vez haciendo los bucles (indicados por WHILE...ENDWHILE anterior) en construcciones de FOR...NEXT, con tal de que se tenga cuidado con la salida desde el exterior. El programa en la figura 5.5 realiza la clasificación en líneas 11Ø a 14Ø.

```
 10 GO SUB 230
100 REM Clasificacion de burbuja
110 LET ptd=8
120 LET sd=0
130 LET ctd=ptd-1
140 LET lop=8
150 LET uop=lop-1
160 IF l$(lop)>=l$(uop) THEN
    GO TO 180
170 LET f$=l$(lop):
    LET l$(lop)=l$(uop):
    LET l$(uop)=f$: LET sd=sd+1
180 LET lop=uop: LET ctd=ctd-1
    IF ctd>0 THEN GO TO 150
190 LET ptd=ptd-1: IF sd=0 THEN
    LET ptd=0
200 IF ptd>1 THEN GO TO 120
210 FOR i=1 TO 8: PRINT l$(i):
    NEXT i
220 STOP
230 DIM l$(8,35)
240 FOR i=1 TO 8: READ l$(i):
    NEXT i
250 RETURN
260 DATA "Sanchez  Carpintero"
270 DATA "Ruiz     Carpintero"
280 DATA "Rivera     Fontanero"
290 DATA "Herrera  Electricista"
300 DATA "BERNAL    Fontanero"
310 DATA "Zorrilla  Carpintero"
320 DATA "Cruz     Electricista"
330 DATA "Garcia    Carpintero"
```

```
Garcia        Carpintero
Bernal        Fontanero
Cruz          Electricista
Herrera       Electricista
Ruiz          Carpintero
Sanchez       Carpintero
Ribera        Fontanero
Zorrilla      Carpintero
```

Figura 5.4.

La salida desde el bucle exterior (p) se hace forzando el valor de p a uno más bajo que el último valor para el que se llevará a cabo el bucle. Recuérdese que éste es el caso de una salida normal, pues el valor del contador se cambia antes de que se pruebe. En Sinclair BASIC, como con la mayoría de las demás versiones, los

```
 10 GO SUB 230
100 REM Clasificacion de burbuja
110 FOR p=7 TO 1 STEP -1:
    LET sd=0:
    FOR c=7 TO 8-p STEP -1
120 IF l$(c+1)<l$(c) THEN
    LET f$=l$(c):
    LET l$(c)=l$(c+1):
    LET l$(c+1)=f$: LET sd=sd+1
130 NEXT c:
    IF sd=0 THEN LET p=0
140 NEXT p
210 FOR i=1 TO 8: PRINT l$(i):
    NEXT i
220 STOP
230 DIM L$(8,35)
240 FOR i=1 TO 8: READ l$(i):
    NEXT i
250 RETURN
260 DATA "Sanchez     Carpintero"
270 DATA "Ruiz        Carpintero"
280 DATA "Ribera       Fontanero"
290 DATA "Herrera  Electricista"
300 DATA "Bernal      Fontanero"
310 DATA "Zorrilla   Carpintero"
320 DATA "Cruz       Electricista"
330 DATA "Garcia     Carpintero"
```

Figura 5.5.

bucles FOR...NEXT son notablemente más rápidos en operación que los que emplean sentencias tales como LET p = p − 1, de modo que lo que podría parecer un método bastante "tosco" de operación resulta ser digno de consideración en lo que respecta a la velocidad. El valor calculado en la tercera sentencia de línea 100 (el bucle c) asegura que no se desperdicie ningún tiempo en comparar registros que se hayan "burbujeado" ya a su lugar correcto en la lista. Una mejora muy pequeña podría efectuarse sustituyendo:

LET sd = sd + 1
por LET sd = 1

con lo que se utiliza la variable sd como un indicador (no hay ninguna necesidad de contar las permutaciones, sino solamente saber si ha tenido lugar una).

Un procedimiento más sofisticado se da al final del capítulo (la clasificación de Shell-Metzner), pero, en lo que respecta a la veloci-

dad, el factor limitador más importante llega a ser la operación del propio intérprete de BASIC. La "carga de trabajo" de traducir cada sentencia en el programa, de encontrar las variables en memoria y de realizar los intercambios es grande en comparación con las mejoras que pueden hacerse utilizando métodos más refinados.

Afortunadamente, la forma en que el intérprete de Sinclair trata a las matrices de caracteres hace que el programa de clasificación de código de máquina sea relativamente fácil de aplicar con el empleo del procedimiento anterior. La matriz que hemos estado utilizando para fines de demostración se retendría en memoria como un bloque único de 280 octetos de longitud (8 registros, cada uno con 35 caracteres). Una vez que se haya localizado el punto de partida, se puede identificar cada registro y el juego de instrucciones del chip del Z80 hace sencillos los procesos de comparar y permutar registros, si se ha adquirido una cierta familiaridad con el código de máquina. En la primera parte del programa siguiente (figura 5.6), una rutina de código de máquina se coloca en el espacio ocupado por una sentencia REM en la primera línea del programa. La sentencia REM tendrá, entonces, un aspecto muy peculiar pero no irá en su detrimento a no ser que trate de aplicarle una edición correctora.

En la segunda parte, comenzando en la línea 240, la información sobre la matriz se introduce en la sentencia REM en la función POKE. Finalmente, en la línea 280, se encuentran las posiciones en memoria de la cadena a clasificar y la cadena de intercambio y se clasifica la cadena. En lo que respecta a las clasificaciones de código de máquina resulta moderadamente rápido (mucho más rápidas que cualquier programa de BASIC). Las instrucciones detalladas son:

1. La rutina puede utilizarse para clasificar cualquier cadena con tal de que se divida en partes de igual longitud. Para una matriz de cadena de dos (o más) dimensiones, éste será automáticamente el caso. En el caso de una matriz unidimensional o de una cadena flexible, la disposición debe llevarse a cabo mediante un programa anterior.
2. A la variable ns se le da el número de cadenas a clasificar (no necesariamente la primera dimensión de la matriz, pues algunas de las cadenas puede estar en blanco y por ello no debe contarse). (Si las cadenas en blanco se incluyen, se colocarán al principio de la lista mediante un proceso de clasificación.) La variable sd es la longitud de cada subcadena; en una matriz bidimensional, ésta será la segunda dimensión. Una cadena de intercambio debe establecerse por una sentencia DIM y ha de ser de la misma longitud que sd. Puede utilizarse cualquier eti-

```
10 REM ..................................
.........................................
.........................................
.........................................
.........................................
.........................................
.........................................
   ( al menos 212 )
 20 DIM l$(24,36): DIM b$(36):
REM DIM b$ lo mismo que la
   segunda de l$
100 FOR i=1 TO 24: LET l$(i)=CHR$
    (90-i): NEXT i
200 LET ns=20: LET sd=36
210 LET p2=INT (ns/256):
     LET p1=ns-256*p2
220 LET p4=INT (sd/256):
     LET p3=sd-256*p4
230 FOR i=1 TO 212: READ p:
     POKE 23759+i,p: NEXT i
240 POKE 23760,p1:
     POKE 23761,p2
250 POKE 23766,p3:
     POKE 23767,p4
280 LET l$(1)=l$(1):
     RANDOMIZE USR 23777:
      LET b$=b$:
       RANDOMIZE USR 23784
290 DIM b$(36): STOP
300 DATA 0,0,0,0,0,0,0,0,0,0
301 REM
302 DATA 0,0,0,0,0,0,0,42,77,92
303 REM
304 DATA 34.210.92.201.42.77.92;
   34,212,92
305 REM
306 DATA 42,208,92,34,217,92,42
   217,92,43
307 REM
308 DATA 34,223,92,34,217,92,12
   5,180,32,1
309 REM
310 DATA 201,175,50,216,92,42,2
   08,92,237,91
311 REM
312 DATA 214,92,43,205,145,93,2
   37,91,210,92
313 REM
314 DATA 25,237,91,214,92,55,63
   229,237,82
315 REM
```

```
316 DATA 235,225,237,83,219,92,
    34,221,92,205
317 REM
318 DATA 74,93,121,183,204,100,
    93,42,223,92
319 REM
320 DATA 43,125,180,32,8,58,216,
    92,183,194
321 REM
322 DATA 244,92,201,34,223,92,4
    2,219,92,195
323 REM
324 DATA 23,93,237,75,214,92,11
    ,121,176,40
325 REM
326 DATA 13,26,190,19,35,40,245
    ,242,96,93
327 REM
328 DATA 1,1,0,201,1,0,0,201,23
    ,7,75
329 REM
330 DATA 214,92,42,219,92,237,9
    1,212,92,237
331 REM
332 DATA 176,237,75,214,92,42,2
    21,92,237,91
333 REM
334 DATA 219, 92,237,176,237,75
    214,92,42,212
335 REM
336 DATA 92,237,91,221,92,237,1
    76,62,1,50
337 REM
338 DATA 216,92,201,76,125,6,16
    ,33,0,0
339 REM
340 DATA 203,57,31,48,1,25,235,
    41,235,16
341 REM
342 DATA 245,201
```

Figura 5.6.

queta (rótulo), pero la matriz a clasificar debe indicarse en la primera parte de la línea 290 y la cadena de intercambio en la segunda.

3. Teclee el programa y grábelo (SAVE) en la cinta antes de ejecutarlo. Sus matrices se grabarán con el programa. Haga la ejecución (RUM) correspondiente y obtenga la salida impresa de la matriz para comprobar el resultado. Si el programa se

deteriorara, o no clasificara de forma adecuada, vuelva a efectuar la carga si fuera necesario y haga una comprobación con respecto a la copia de forma muy cuidadosa. Ha de cerciorarse de que tiene el número correcto de caracteres en la sentencia REM. ¡El programa se comprobó a fondo antes de su publicación!

4. Cuándo esté trabajando de forma adecuada, conserve el código de máquina solamente utilizando:

   SAVE "clasificación"CODE23760,212

   (No "añada uno al azar" al número de octetos, a no ser que haga también lo mismo en la sentencia REM). Puede verificarse (VERIFY" CODE es suficiente). Si necesita una clasificación en otro programa, sólo tendrá que introducir por el teclado la sentencia REM (*debe* ser la primera línea) y cargar LOAD" CODE. En efecto, el código de máquina se funde (MERGE) en el nuevo programa. El nuevo programa debe contener los equivalentes de líneas 200,210,220,240,250, y 280. Pueden estar en cualquier lugar en el programa. La línea 250 no se necesita si ha seguido este plan.

El tiempo tardado variará dependiendo de si la matriz ya está parcialmente clasificada. En una prueba, 300 cadenas aleatorias se clasificaron en 20 segundos.

DOBLE BURBUJA

En todos los programas de clasificación, expuestos hasta ahora, la cadena completa se ha utilizado para fines de comparación. Ello significaría, por ejemplo que

Martínez	Fontanero
y Martínez	Carpintero

se intercambiarían porque "Carpintero" es menor que "Fontanero" (alfabéticamente hablando, por supuesto). Si comenzáramos la comparación más adelante en la cadena, ignorando nombres y concentrándose plenamente en las profesiones, todos los carpinteros se elevarían por encima de los electricistas y así sucesivamente. El programa en la figura 5.7 sigue una clasificación por otra, primero en la cadena completa y luego, por profesión. El resultado es que los carpinteros van a la parte superior; pero están en orden alfabético, como lo están todas las demás profesiones. Ello tiene aplicaciones obvias. Para ahorrar espacio, la clasificación se escribe como una subrutina y la variable st se "pasa" a ella. Cuando se utiliza esta técnica, recuérdese clasificar primero el factor menos importante.

```
  10 GO SUB 230
100 REM Doble calsificacion de
    burbuja
110 LET st=1: GO SUB 150:
     LET st=16: GO SUB 150
120 FOR i=1 TO 8: PRINT l$(i):
    NEXT i
130 STOP
150 FOR p=7 TO 1 STEP -1:
    LET sd=0:
    FOR c=7 TO 8-p STEP-1
160 IF l$(c+1,st TO )<
    l$(c,st TO ) THEN
    LET f$=l$(c):
    LET l$(c)=l$(c+1):
    LET l$(c+1)=f$: LET sd=sd+1
170 NEXT c:
180 NEXT p: RETURN
230 DIM l$(8,35)
240 FOR i=1 TO 8: READ l$(i):
    NEXT i
250 RETURN
260 DATA "Sanchez   Carpintero"
270 DATA "Ruiz      Carpintero"
280 DATA "Ribera     Fontanero"
290 DATA "Herrera  Electricista"
300 DATA "Bernal     Fontanero"
310 DATA "Zorrilla  Carpintero"
320 DATA "Cruz     Electricista"
330 DATA "Garcia    Carpintero"
```

```
Garcia         Carpintero
Ruiz           Carpintero
Sanchez        Carpintero
Zorrilla       Carpintero
Cruz           Electricista
Herrera        Electricista
Bernal         Fontanero
Ribera         Fontanero
```

Figura 5.7.

5.3 Búsqueda a través de matrices clasificadas

Hasta ahora, hemos buscado un registro particular explorando profundamente a través de la lista desde el principio al final (una búsqueda en serie). Es evidente que ello puede mejorarse de alguna forma si la lista ya está clasificada. Un método bastante sencillo se conoce como búsqueda binaria. Dos indicadores se estable-

cen para empezar al principio y al final de la lista. Suponiendo que ninguno de los dos ha incidido en el elemento de datos requerido, la lista se divide en dos, tan pronto como sea posible. Si el indicador de marca intermedio no ha indicado el objetivo, se realiza una comprobación para ver en cuál de sus lados está el objetivo (ello sólo es posible con una lista ordenada). Uno de los marcadores extremos está ahora "arrancado" y colocado en medio, con lo que se divide en dos el objeto de la búsqueda. Este proceso continúa hasta que uno u otro de los marcadores queda apartado de forma individual, en cuyo caso la búsqueda ha sido infructuosa o hasta que uno de ellos "da en el blanco".

En principio, éste es un método sencillo pero, como en muchos procesos computarizados, ha de tenerse bastante cuidado en cerciorarse de que son correctas las condiciones de salida (EXIT) y de que se prueban en la secuencia adecuada. Los principios y los finales de listas también exigen, a veces tratarse con precaución, como veremos más adelante. Dado que resolvimos estos problemas, la búsqueda binaria es rápida, requiriéndose solamente 10 repeticiones (a lo sumo) una lista de 1000 elementos de datos.

Supóngase que la lista se ha clasificado en orden ascendente, semejante a una lista alfabética. Primero se prueban el principio y el final de la lista y luego se introduce un bucle. El marcador intermedio se calcula y se compara con respecto al objetivo, primero para ver si es igual y luego para determinar si es más grande. El resultado de la segunda comprobación nos dice si ha de desplazarse el marcador a la derecha o a la izquierda. Una vez desplazado, se mide la separación entre los dos extremos y, si la hubiere, el bucle es objeto de salida. De cualquier otro modo, el proceso se repite. Obsérvese que, de hecho, no probamos los marcadores extremos cada vez. Una vez probados permanecen en este estado. El diagrama puede servir para aclararlo. Muestra las posibilidades si comenzamos con una lista de 12 datos elementales. Cada línea presenta todos los marcadores posibles en cualquier etapa y el objeto se encamina, como a través de un embudo, hacia una de las columnas.

1											12
1					6						12
1		3			6			9			12
1	2	3	4		6	7		9	10		12
			4	5	6	7	8	9	10	11	12

Obsérvese que algunas columnas son más largas que otras. Puesto que la lista se divide por la mitad en cada repetición, el nú-

mero máximo de pasos dados viene dado por la más baja potencia de dos que sea más grande que el número de elementos de datos. Por ejemplo, puesto que 2 a la potencia de 4 es igual a 16, el número máximo de pasos con 12 elementos de datos es 4.

```
  PROC Búsqueda binaria
  LET encontrado = n
  LET marca _ derecha = Número _ de _ elementos _ de _
datos
  LET marca _ izquierda = 1
  IF 1$ (marca _ derecha) = búsqueda$ THEN LET encontrado
= y: EXIT
  IF 1$ (marca _ izquierda) = búsqueda$ THEN LET encontrado
= y : EXIT
  REPEAT
    LET (marca _ media) = INT (marca _ izquierda + marca _
      derecha/2)
    IF 1$ (marca _ media) = búsqueda$ THEN LET encontrado
      = y : EXIT
    IF 1$ (marca _ media) > búsqueda$ THEN
      LET marca _ derecha = marca _ media
    ELSE
      LET marca _ izquierda = marca _ media
    ENDIF
    ENDREPEAT ON marca _ derecha—marca _ izquierda = 1
  PRINT (mensaje adecuado)
  ENDPROC
```

En el programa (figura 5.8), el bucle de búsqueda ocupa las líneas 140 a 180 inclusive. La clasificación de burbuja en las líneas 210 a 230 muestra (en la línea 230, segunda sentencia) un método alternativo de prueba de si la lista está en orden. La lista de variables y la salida impresa se ilustran en la figura 5.9.

5.4 La clasificación de Shell—Metzner

La rutina trabaja en el modo operativo de comparación continuada, pero los pares no son contiguos. El "offset" de variable da la distancia entre las cadenas a comparar. Con una lista de nueve variables como en el programa, el "offset" comienza en 4. En la primera pasada, se compararán los números 1 y 5, luego 2 y 6, 3 y 7 y así sucesivamente. El resultado de una pasada es un conjunto de listas intercaladas, cada una en orden correcto, con los elementos de cada lista desplazándose en el valor de "offset". Este último se divide por la mitad antes de cada pasada, por lo que au-

```
 10 DIM l$(12,12): GO SUB 200
100 REM Busqueda binaria *******
110 LET y=1: LET n=0: LET lm=1:
    LET rm=12: LET encontrado=n:
    LET q=0: LET s=0
120 IF l$(rm, TO ql)=q$ THEN
    LET q=rm: LET encontrado=y:
    GO TO 190
130 IF l$(lm, TO ql)=q$ THEN
    LET q=lm: LET encontrado=y:
    GO TO 190
140 LET mm=INT ((rm+lm)/2):
    LET s=s+1
150 IF l$(mm, TO ql)=q$ THEN
    LET q=mm: LET encontrado=y:
    GO TO 190
160 IF l$(mm)<q$ THEN LET lm=mm
170 IF l$(mm)>q$ THEN LET rm=mm
180 IF rm-lm<>1 THEN GO TO 140
190 PRINT q$
    ("NO" AND NOT encontrado)
    ;"Encontrado ";
   ((" en ";s;" STR$ q) AND encon
    trado)
  ;" en " s;" pasos"
195 STOP
200 FOR i=1 TO 12: READ l$(i):
    NEXT i:
    INPUT "Solicitud busqueda:
    "q$:
    LET ll=LEN q$
210 FOR p=11 TO 1 STEP -1
    LET sd=0:
    FOR c=11 TO 12-p STEP -1
220 IF l$(c+1)<l$(c) THEN
    LET f$=l$(c):
    LET l$(c)=l$(c+1):
    LET l$(c+1)=f$: LET sd=sd+1
230 NEXT c: LET p=p*(sd<>0):
    NEXT p
240 FOR i=1 TO 12: PRINT l$ (i):
    NEXT i: PRINT
250 RETURN
310 DATA "Juan","Maria","Sara"
320 DATA "Raquel","Ana","Lucia"
330 DATA "Laura","Clara","Juana"
340 DATA "Eva","Luisa","Ines"
```

Figura 5.8.

menta el número de comparaciones por pasada, pero cuando el "offset" alcanza una el resultado es una lista clasificada. La clasificación de Shell implica menos comparaciones que la de burbuja, pero la ganancia no es muy grande cuando se utiliza BASIC (figura 5.10)

```
Ana
Clara
Ines
Eva
Juan
Juana
Laura
Lucia
Luisa
Maria
Raquel
Sara


Eva encontrado en 4 en 3 pasos.




Lista de variables
------------------

l$( )          Matriz
i£             Contador
q$             Cadena de busqueda
ql             Longitud cadena de busqueda
p£             Pasadas en clasificacion
sd             Permutaciones hechas en clasificaciones
c£             Comparaciones en clasificaciones
f$             Intercambio en clasificaciones
y              Verdadero -1
n              Falso -Ø
lm             Marcador izquierdo
rm             Marcador derecho
encontrado     Indicador
q              Posicion encontrada
s              Numeros de pasos
mm             Marcador medio
```

Figura 5.9.

```
 10 LET nl=9: GO SUB 230
    LET offset=INT (nl/2)
100 REM Clasif Shell-Metzner
110 IF offset<1 THEN GO TO 210
120 LET comps=nl-offset
130 FOR p=1 TO comps:
     FOR c=p TO 1 STEP -offset
140 IF l$(c+offset)<l$(c) THEN
     LET f$=l$(c)
      LET l$(c)=l$(c+offset):
       LET l$(c+offset)=f$
150 NEXT c: NEXT p:
     LET offset=INT (offset/2):
      GO TO 110
210 FOR i=1 TO nl: PRINT l$(i):
     NEXT i
220 STOP
230 DIM l$(nl,15)
240 FOR i=1 TO nl: READ l$(i):
     NEXT i
250 RETURN
260 DATA "Sanchez"
270 DATA "Ruiz"
280 DATA "Ribera"
290 DATA "Herrera"
300 DATA "Bernal"
310 DATA "Zorrilla"
320 DATA "Cruz"
330 DATA "Garcia"
340 DATA "Abajo"
```

Figura 5.10.

5.5 Ficheros de índice

Para métodos de búsqueda más eficaces, es necesario conocer más sobre el contenido de la lista. Un fichero de índice es un fichero que contiene breves referencias a cada elemento de datos en la lista por lo que puede acortarse la búsqueda. Se suele actualizar de forma continua, por lo que cada vez que se altere, añade o suprime un elemento de datos, se cambiarán dos registros: el registro en el fichero principal y la referencia en el índice. Ello requiere que sea efectiva mucha organización, pero con listas largas puede merecer la pena. Una forma simplificada de indexación implicaría mantener punteros para los primeros elementos de datos que comienzan con A,B,C,... y así sucesivamente, de modo que las marcas al principio de una búsqueda podrían establecerse para abarcar un alcance menos amplio. Una versión de este último se pondrá de manifiesto en el capítulo siguiente.

6 MANTENIMIENTO DE SUS FICHEROS

En este capítulo se darán tres programas bastante grandes. Cada uno es adecuado para mantener ficheros de información, pero presentan, de forma gradual, medios más sofisticados y, por supuesto, utilizan técnicas más complicadas. Los programas se han desarrollado con el empleo de una estructura básica común, con algunas partes intercambiables. Algunas de las rutinas en otras partes del libro pueden, con poca o ninguna adaptación, añadirse a los programas en este capítulo, por lo que podrá ser capaz de concebir un producto final que sea adecuado para sus propios fines. Como ocurre en muchas construcciones de esta naturaleza, será fácil su modificación o corrección porque está constituido a partir de módulos normalizados.

6.1 El diseño

La misma estructura desarrollada para los programas de la agenda y del listín telefónico se utilizará en los dos primeros programas, en los cuales se emplean registros de longitud fija almacenados en matrices bidimensionales. La rutina de menú es idéntica a la mostrada en el programa del listín telefónico al final del capítulo 4, pero se necesita una subrutina de utilidad adicional en la línea **800** (figura 6.1), aunque, de hecho, no se utilizará hasta el segundo programa. Con esta excepción, todas las rutinas por debajo de la línea **1000**, junto con las ubicadas en las líneas **6000** y **7000**, se han copiado del programa del listín telefónico, con sólo un pequeño,

```
800 REM Mayusculas *****
810 IF ic>0 AND ic<27 THEN
     LET ok=y: RETURN
820 LET ok=n:
     PRINT AT 19,0; FLASH 1;
    "Entrada no valida-Sirvase pro
"    bar de nuevo"
830 GO SUB 500:
     PRINT AT 19,0;TAB 31;:
      RETURN
```

Figura 6.1.

pero importante cambio —la sentencia DIMENSION en la línea 6040 (los detalles más adelante)—. El diseño deja algunas zonas libres para las adiciones (podría incorporarse fácilmente la entrada de pantalla concebida para el programa de la agenda de direcciones).

6.2 Lista de nombres

El primer programa almacena una simple línea de apellidos. Cada registro tiene solamente un campo, con 30 caracteres de longitud, pero no hay ningún motivo por el que no deba adaptarse para contener varios campos en la misma forma utilizada en los programas anteriores. El punto principal a considerar es el método mediante el cual los registros se mantienen en orden alfabético, sin utilizar rutinas de clasificación.

6.3 La lista encadenada

En una "caza del tesoro", cada pista nos dice en donde encontrar la siguiente, de forma sucesiva, hasta que alcancemos el destino final. Una lista encadenada, o enlazada, está organizada de forma similar, sustituyéndose las pistas por punteros numéricos. Los propios registros se mantienen en una matriz. El orden real del almacenamiento suele ser aquel en el que se introduce, que puede ser completamente aleatorio, pero los punteros determinan cómo serán objeto de lectura. En la jerga de las computadoras, éste se

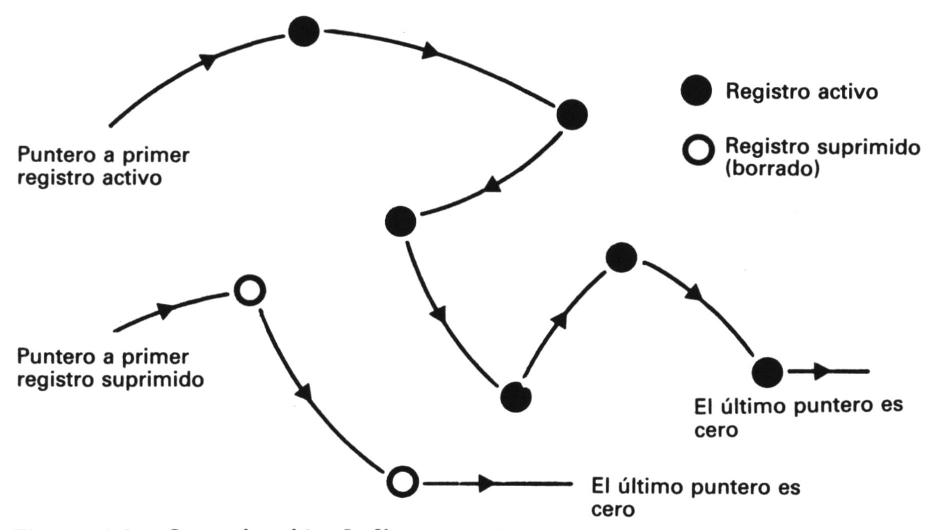


Figura 6.2.—Organización de lista encadenada.

suele denominar el orden lógico (en el ámbito de las computadoras, el término “lógico” no tiene relación, a veces, con un razonamiento correcto, sino que se refiere a la forma en la que vemos algo como opuesto a la forma en que se observa por la computadora). La diferencia entre las pistas para el descubrimiento del tesoro y los punteros en una lista encadenada es que, a medida que se suprimen y añaden registros, los punteros tienen que ajustarse para asegurar que, cualquiera que sea el orden físico, el orden lógico siempre será correcto.

En el programa se mantienen dos cadenas de punteros. La cadena principal, indicada en la figura 6.2 por círculos rellenados, conecta registros que son “activos”; esto es, todavía requerido por el usuario. Los circulos vacíos indican registros que han sido borrados. Para evitar la tergiversación, los registros suprimidos no son inmediatamente borrados de la memoria, sino añadidos al principio de la segunda lista. Más adelante, cuando se añaden otros registros, serán recubiertos por escritura. En la figura 6.3 se muestra el proceso de añadir un registro a una lista de la que se han suprimido dos registros. El nuevo registro recubre al primero en

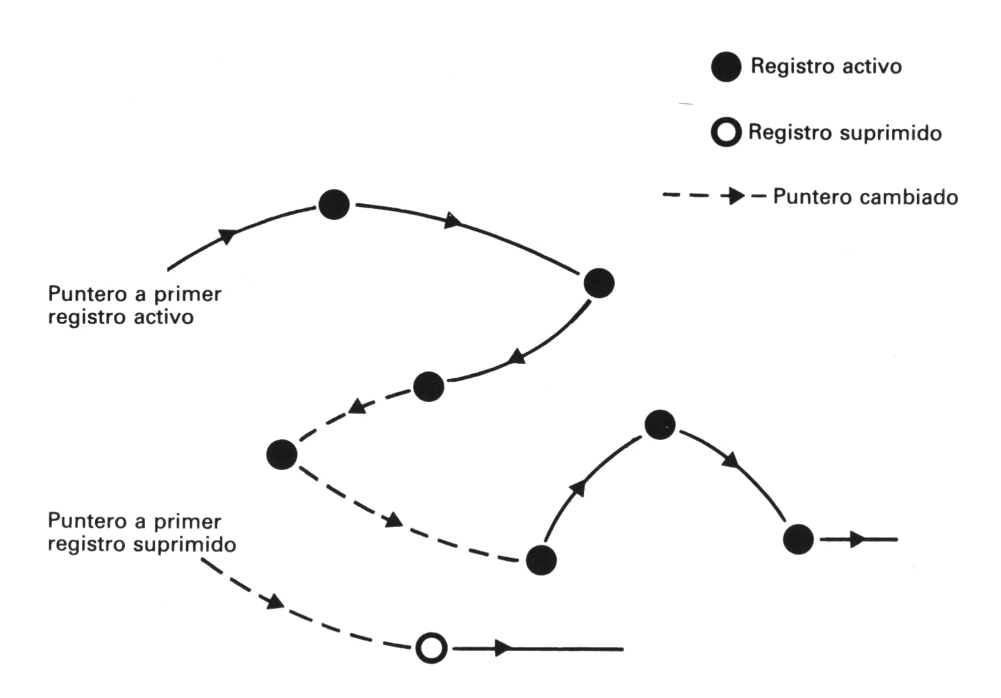


Figura 6.3.—Lista encadenada—adición de un registro.

la segunda cadena y luego se cambiarán dos conjuntos de punteros. Los registros y los punteros se almacenas en matrices en el programa, por lo que la línea 6040 debe leerse ahora :

```
6040 IF ok THEN DIM l$(100,30):
     LET rc = 0 : DIM r$(30) :
     DIM p(100)
```

```
1000 REM Hacer una nueva entrada ****
1010 PRINT AT 2,0;"Nombre";:
     LET ec=0
1020 IF rc=30 THEN BEEP .2,40:
      PRINT AT 20,0;TAB 31;
     AT 11,8; FLASH 1;
    "El fichero esta lleno M":
     GO SUB 500: RETURN
1030 PRINT AT 20,2; INVERSE 1;
    "Introduzca detalles luego"
    : INPUT "Nombre:"r$
1040 PRINT AT 20,0;TAB 31:
     PRINT AT 3+ec,0;r$:
     GO SUB 300
1050 IF modifica THEN RETURN
1060 IF NOT ok THEN
     PRINT AT ec+3,0;TAB 31:
     GO TO 1030
1070 LET rc=rc+1: IF pfs=0 THEN
     LET l$(rc)=r$: LET cr=cr:
     GO TO 1090
1080 LET cr=pfs: LET l$(cr)=r$:
     LET pfs=p(cr)
1090 IF pfr=0 THEN LET pfr=cr:
      LET nr=0: GO TO 1130
1100 LET pr=pfr: LET nr=pfr:
      IF l$(nr)>r$ THEN
       LET pfr=cr: GO TO 1130
1110 IF l$(nr)<=r$ THEN
      LET pr=nr: LET nr=p(nr):
       IF nr<>0 THEN GO TO 1110
1120 LET p(pr)=cr
1130 LET p(cr)=nr
1140 IF modifica THEN RETURN
1150 PRINT AT 20,4; FLASH 1;
    "Otra entrada (s/n)"
1160 PAUSE 0: LET a$=INKEY$:
     IF a$="s" OR a$="S" THEN
     LET ec=ec+1:
     LET ec=ec*(ec<16):
     GO TO 1020
1200 RETURN
```

Figura 6.4.

Con ello se establece el número máximo de registros en 100. En un Spectrum de 48K se podría almacenar más de 1000 de esa longitud, pero ha de tenerse presente que hay que cambiar las dimensiones para 1$ y p.

El pseudocódigo siguiente da una descripción exacta del proceso de adición de un registro. Para poder programar eficazmente, se utilizan dos punteros especiales, uno para el primer registro en la cadena principal (puntero _ primer _ registro) y uno similar para los registros suprimidos denominado puntero _ primer _ registro. Obsérvese que el procedimiento ha de aprovecharse de varias posibilidades: la lista principal puede estar completamente vacía, puede haber, o no, registros suprimidos, etc. Para comprender lo que está sucediendo ha de hacerse una primera lectura de la descripción pasando superficialmente por lo que no sean los comentarios (encerrados entre dobles guiones) y volviendo luego a examinar cada subproceso de forma sucesiva. La parte más frecuentemente utilizada es la búsqueda en serie a lo largo de la cadena para encontrar la posición correcta para un nuevo registro (la sección WHILE...ENDWHILE hacia el final. En el programa, ocupa las líneas 1070 y 1130 (figura 6.4)

```
PROC Añadir un nuevo registro
//Introducir registro y poner punteros de espacio//
LET conteo _ reg = conteo _ reg + 1
IF punteo _ primero _ espacio = 0 THEN
  //Añadir al final de lista//
  LET reg _ actual = conteo _ reg
  LET 1$ (reg _ actual) = reg$
ELSE
  //Añadir al comienzo de la cadena de registros suprimidos//
  LET reg _ actual = punto _ primer _ espacio
  LET 1$ (reg _ actual) = reg$
  LET punto_ primer _ espacio =p (reg _ actual)
ENDIF
//Ahora poner los punteros de registros//
IF punto _ primer _ reg =0 THEN
  //este es el primer registro//
  LET punto _ primer _ reg = reg _ actual
  LET sig _ reg = 0
ELSE
  //Poner los punteros//
  LET reg_ prev = punto _ primer _ reg
  LET sig _ reg = punto _ primer _ reg
  IF 1$ (sig _ reg) > reg$ THEN
    //Insertar en cabeza de lista//
```

```
        LET punto _ primer _ reg = reg _ actual
        ELSE
        //Búsqueda para encontrar posición correcta//
        WHILE 1$ (sig _ reg) < reg$ AND p (sig _ reg) < > 0
            LET reg _ prev = sig _ reg
            LET sig _ reg = p (sig _ reg)
        ENWHILE
        //Ajustar puntero "a"//
        LET p (reg _ prev) = reg _ actual
      ENDIF
      ENDIF
    //Ajustar puntero "desde"//
    LET p (reg _ actual) = sig _ reg
    ENDPROC
```

En la figura 6.5 se muestra el estado de los punteros en el fichero, primero después de haber añadido 12 registros y luego, después de haber suprimido dos de ellos. Los dos punteros especiales se visualizan por encima de la lista, con el índice del registro a la izquierda y el puntero correspondiente a la derecha. Comenzando por los registros indicados es posible seguir las cadenas en orden lógico y confirmar la corrección de la organización.

La segunda cadena merece una atención especial. A diferencia con la primera, los registros que se han suprimido no se añaden en la posición correcta (no habría ningún punto). Por el contrario, simplemente se colocan en la cabeza de la lista. Cuando se reutilizan, se vuelven a tomar desde la parte superior. Esta estructura se conoce como una "pila" y es, esencialmente, la misma que la utilizada para mantener el registro de los números de línea de RETURN por medio del intérprete de BASIC cuando se utilizan subrutinas múltiples en un programa. También puede ver una pila denominada una cola LIFO (último en entrar, primero en salir).

HALLAZGO DE UN REGISTRO

Como en otros programas, la rutina de búsqueda actúa con el número de caracteres introducidos por el usuario y permite la continuación si el primer registro encontrado no es satisfactorio. Se trata de una búsqueda en serie, pero prosigue a lo largo de la cadena en un orden lógico en oposición al orden del índice, comenzando siempre en el registro apuntado a punto _ primer _ reg. Se termina tan pronto como encuentre un registro cuya primera parte sea alfabéticamente más grande que la cadena dada. El bucle de búsqueda se encuentra en la línea 4050 (figura 6.6) y tiene también salvaguardas para evitar sobrepasar el final de la lista; esta es la razón para hacer cero el último puntero. Es una rutina complicada,

```
       Lista alfabetica

   pfr=4                pfs=Ø
    1 Trucha S                 3
    2 Salmon B                 12
    3 Cebra K                  Ø
    4 Armadillo M              5
    5 Oso hormiguero P         11
    6 Leon K                   9
    7 Leopardo C               8
    8 Mofeta M                 2
    9 Lagarto S                8
   1Ø Jaguar C                 7
   11 Elefante P               1Ø
   12 Foca I                   1
```

```
       Lista alfabetica

   pfr=5                pfs=6
    1 Trucha S                 3
    2 Salmon B                 12
    3 Cebra K                  Ø
    4 Armadillo M              Ø
    5 Oso hormiguero P         11
    6 Leon K                   4
    7 Leopardo C               9
    8 Mofeta M                 2
    9 Lagarto S                8
   1Ø Jaguar C                 7
   11 Elefante P               1Ø
   12 Foca I                   1
```

Figura 6.5.

pero compacta, que se cuidará de algunas posibilidades fácilmente pasadas por alto; v.g. dos nombres idénticos, pero el usuario no debe introducir más de 30 caracateres en la cadena de búsqueda. El indicador (variable found-variable encontrada) se utiliza para retornar un mensaje a otras rutinas que hacen uso de la búsqueda. Esencialmente, el mismo bucle se emplea cuando la lista está impresa-línea 5020 (fig. 6.7). Estos bucles no pueden sustituirse por otros de FOR...NEXT, porque el tamaño del paso es desigual. Para la búsqueda en sí misma, no es necesario mantener una anotación del anterior registro "visitado" (variable pr) pero se utilizará en las rutinas de borrado y de modificación.

```
4ØØØ REM Encontrar una entrada *******
4Ø1Ø PRINT AT 2Ø,1; INVERSE 1;
     "Introduzca el nombre a en
      contrar a continuacion"
4Ø2Ø INPUT"Nombre:";s$:
      LET sl=LEN s$: LET cr=pfr:
       LET pr=Ø: LET encontrada=n
4Ø3Ø PRINT AT 2Ø,Ø;TAB 31;
     AT 2Ø,1Ø; FLASH 1;
     "BUSQUEDA"
4Ø5Ø IF cr<>Ø THEN LET encontra
     da=s:
     IF s$<>l$(cr, TO sl) THEN
      LET pr=cr: LET encontrada
      =n:
      LET cr=p(cr): GO TO 4Ø5Ø;
      1Ø*(s$<l$(pr, TO sl))
4Ø6Ø IF NOT encontrada THEN
     PRINT INVERSE 1;AT 8,3;
     "Ninguna coincidencia (mas)
       encontrada"
  :   GO TO 419Ø
4Ø7Ø PRINT AT 8,11;"ENCONTRADA"
    AT 2Ø,Ø;TAB 31
4Ø8Ø PRINT AT 11,Ø; INVERSE 1;
    l$(cr);
4Ø9Ø PRINT AT 2Ø,4; FLASH 1;
    "Continuar busqueda (s/n)?"
41ØØ PAUSE Ø: IF INKEY$="S" OR
    INKEY$="s" THEN LET pr=cr:
    LET encontrada=n: LET cr=
    p(cr):
    GO TO 4Ø5Ø
419Ø GO SUB 5ØØ: RETURN
```

Figura 6.6.

```
5ØØØ REM Imprimir la lista
5Ø1Ø LET cr=pfr: LET i=3
5Ø2Ø IF cr<>Ø THEN
     PRINT AT i,Ø;l$(cr):
     LET cr=p(cr): LET i=i+1:
      IF i<18 THEN GO TO 5Ø2Ø
5Ø4Ø PRINT AT i,Ø;TAB 31;" ":
     GO SUB 5ØØ
5Ø5Ø LET i=3: IF cr<>Ø THEN
    GO TO 5Ø2Ø
5Ø6Ø RETURN
```

Figura 6.7

```
2000 REM Suprimir una entrada *****
2010 GO SUB 4000: IF NOT encon
     trada
     THEN PRINT AT 11,0;TAB 31;.
      GO SUB 500: RETURN
2020 PRINT AT 20,4; FLASH 1;
     "Pulse ' ' para borrar ";
     AT 21,4;
   "cualquier otra tecla para
    retornar";
2030 PAUSE 0:
      PRINT AT 20,0;TAB 31;
     AT 21,0;TAB 31;
2040 IF INKEY$<>"s"
      AND INKEY$<>"S" THEN RETURN
2050 LET rc=rc-1: LET nr=p(cr):
       LET pss=pfs: LET pfs=cr:
       LET p(cr)=pss
2060 IF pr=0 THEN LET pfr=nr:
       GO TO 2080
2070 LET p(pr)=nr
2080 GO SUB 500: RETURN
```

Figura 6.8.

SUPRESIÓN (FIGURA 6.8)

En primer lugar, se realiza una búsqueda y se obtiene la confirmación adecuada. El borrado real se efectúa en las líneas 2050 a 2070 y se resume a continuación.

```
PROC Supresión de un registro
//Introducción en la pila de registros suprimidos//
LET punto _ segundo _ espacio = punto _ primer _ espacio
LET punto _ primer _ espacio = reg _ actual //El que ha de
  suprimirse//
LET p (punto _ primer _ espacio) = punto _ segundo _ espacio
//Ahora ajuste los punteros en lista principal//
IF reg _ prev = 0
  //Vamos a borrar el único registro//
  LET punto _ primer _ reg = 0
ELSE
  //Eludir el registro suprimido//
  LET p (reg _ prev) = sig _ reg
ENDIF
LET conteo _ reg = conteo _ reg − 1
ENDPROC
```

```
3000 REM Modificar una entrada
3010 GO SUB 4000:IF NOT encon
     trada THEN PRINT AT 11,0;
     TAB 31;:GO SUB 500: RETURN
3020 PRINT AT 20,2; INVERSE 1;
    "Introduzca enmienda ahora"
     : INPUT "Nombre",",r$:
     PRINT AT 13,0; INVERSE 1;r$
     : GO SUB 300:
     IF NOT ok THEN GO TO 3020
3030 PRINT AT 20,0;TAB 31;
     AT 20,4; FLASH 1;
    " Pulse 'y' para modifi
     car";
    AT 21,4;
    "cualquier otra tecla para re
     tornar";
3040 PAUSE 0:
     PRINT AT 20,0;TAB 31;
     AT 21,0;TAB 31;
3050 IF INKEY$<>"Y"
     AND INKEY$<>"y" THEN RETURN
3060 GO SUB 2050; REM supresion
3065 LET cambio=y: GO SUB 1070:
     LET cambio=n: REM adicion
3070 PRINT AT 11,0; INVERSE 1;
     r$;AT 13,0;
     INVERSE 0;TAB 31;" "
3080 GO SUB 500: RETURN
```

Figura 6.9.

ALTERACIONES (FIGURA 6.9)

La rutina para modificar una entrada necesita pocos comentarios, puesto que se construye a partir de lo ya descrito. La comprobación en la línea 3010 asegura que una vez que se haya continuado la búsqueda más allá del último registro de coincidencia, no se puede intentar ninguna modificación (porque los punteros serán incorrectos). La modificación se toma como una supresión seguida por una adición a la lista, no realizándose nada hasta que todo se haya comprobado. El hecho de trabajar de esta forma simplifica mucho el proceso de ajustar los punteros, pues pueden utilizarse rutinas anteriormente escritas.

LA LEY DE LA FATALIDAD Y LA DEPURACIÓN

La ley de la fatalidad podría enunciarse en la forma de “lo que tenga que suceder, sucederá”. Es muy importante tenerlo pre-

sente en la concepción de programas, porque en las etapas iniciales de diseño resulta demasiado fácil limitar la atención a las situaciones ideales y olvidarse de hacer frente a las no habituales. ¿Qué sucede, por ejemplo, cuando dos registros idénticos se añaden al fichero? ¿Se encarga el programa de las adiciones y de las supresiones correctas al final/principio del fichero? ¿Qué sucede cuando el fichero está completo? Durante el diseño inicial y la programación, es necesario estudiar estos problemas y probar el comportamiento del programa. Si las cosas no funcionan adecuadamente, habremos de encontrar un error ("gazapo") en la programación. Dos métodos son posibles si encuentra uno. Si no es demasiado grave y (o) tiene efectos interesantes, puede escribirle en la descripción del programa, con lo cual debe denominarse una característica ("una característica interesante de este programa es que ..."). De forma alternativa, y con mayor seriedad, el programa ha de modificarse para retener o depurar el error. Si encontrase una anomalía que diera lugar a un importante rediseño, habrá de tener mucho más cuidado cuando planifique su siguiente esfuerzo (este párrafo *no* está escrito como un reto para encontrar las formas de hacer confusos los programas en este capítulo).

Una característica de último programa, que es inherente al diseño, es que la lista está desequilibrada. Tenemos un puntero al principio de la lista y ése es el único punto de partida. Por consiguiente, cuando se realizan búsquedas (y prácticamente cualquier procedimiento implica una), cuanto más lejos está el objetivo a través de la lista, tanto mayor es el tiempo que dura. Ello no es demasiado grave si ha de almacenarse un pequeño número de registros grandes, pues el tiempo de acceso de registro a registro es el mismo, cualquiera que sea la longitud del registro, pero si se mantienen grandes números de registros pequeños puede resultar tediosa la espera de los registros al final de la lista. El siguiente programa supera este problema, en alguna medida, a expensas de un programa algo más largo y más complejo.

6.4 Lista encadenada con punteros alfabéticos

En la figura 6.10 se ilustra cómo se controla esta lista. Se trata de un desarrollo del programa anterior, con el empleo del mismo sistema para las dos cadenas de registros activos y suprimidos respectivamente. Asimismo, una adición de 26 punteros indican el primer registro para cada letra del alfabeto. El conjunto de punteros alfabéticos constituye un fichero de índice rudimentario y permite

que las búsquedas se limiten a aquellos registros con la letra inicial correcta. El diagrama muestra la situación para la lista:

BARRY
BRENDA
FIONA
FRANCÉS
HARRIET
HENRY
JOHN

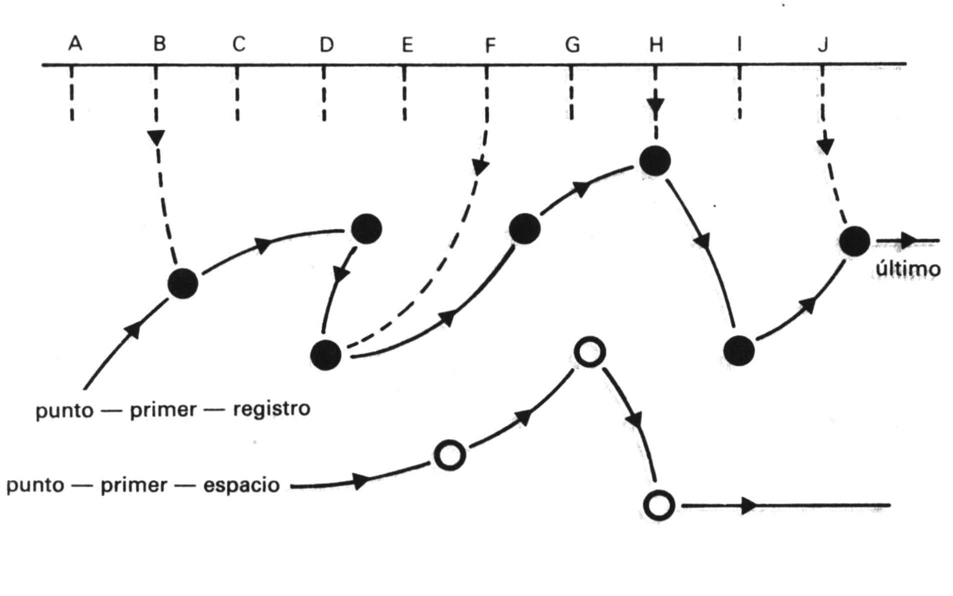


Figura 6.10—Lista encadenada con punteros alfabéticos.

con tres registros suprimidos. Los punteros alfabéticos sin ningún registro correspondiente son puestos a uno o permanecen en cero. En el programa, se encontró un puntero adicional de utilidad para evitar un desbordamiento en los procesos de búsqueda, por lo que la línea 6040 tiene, ahora, la expresión:

```
6040  IF ok THEN DIM 1$(100,30):
      LET rc =0 : DIM r$(30) :
      DIM p(100) : DIM a(27)
```

El índice para el puntero alfabético de un registro se calcula restando 64 del CODIGO (valor ASCII) del primer carácter. La subrutina adicional para comprobar este valor se insertó en la línea 800 (ver figura 6.1). Exige que todos los registros hayan de comenzar con una letra mayúscula. Un procedimiento alternativo habría sido realizar automáticamente la conversión, de una forma similar a la utilizada en el programa de conteo de letras dado en un capítulo anterior, pero seguiría siendo necesario los signos de puntuación o los numerales como indicadores.

6.5 Inserción de un registro

La rutina correspondiente a esta inserción (figura 6.11) es bastante larga; las líneas 1070 a 1190, con una subrutina adicional en 1300, que se emplea por otras rutinas de búsqueda en el programa. La versión completa en pseudocódigo es larga por lo que solamente se dará un bosquejo. El registro se inserta primero en exactamente la misma forma que antes y se calcula el valor de ic (código inicial). Si el puntero alfabético correspondiente a ese valor es cero (sin entrada anterior para la misma letra), el programa busca primero hacia adelante para encontrar el siguiente puntero alfabético que no es igual a cero, lo que da también el siguiente registro en secuencia (ésta es la razón para el vigésimo séptimo puntero alfabético). A continuación, se utiliza la rutina de contrabúsqueda en la línea 1300. Su acción es buscar primero hacia atrás para encontrar un puntero no cero y luego hacia adelante, para encontrar el último en esa parte de la cadena. Imaginemos la inserción de DAVID en la lista anterior. En primer lugar, retrocederíamos a BARRY (no hay ningún nombre que comience con c) y luego hacia adelante hasta BRENDA. Con esta operación se obtiene el registro anterior en la cadena completa. Si el puntero alfa-

```
1000 REM Hacer una nueva entrada ****
1010 PRINT AT 2,0;"Nombre";:
     LET ec=0
1020 IF rc=100 THEN BEEP .2,40:
      PRINT AT 20,0;TAB 31;
     AT 11,8; FLASH 1;
     "Fichero esta lleno":
    GO SUB 500: RETURN
1030 PRINT AT 20,2; INVERSE 1;
   "Introducir detalles ahora"
  :  INPUT "Nombre:";r$
1040 PRINT AT 20,0;TAB 31;
     PRINT AT 3+ec,0;r$
```

```
1050 GO SUB 600: IF NOT ok THEN
     PRINT AT ec+3,0;TAB 31:
     GO TO 1030
1060 LET ic=CODE r$(1)-64:
     GOSUB 800: IF NOT ok THEN
     PRINT AT ec+3,0;TAB 31:
     GO TO 1030
1070 LET rc=rc+1: IF pfs=0 THEN
     LET l$(rc)=r$: LET cr=rc:
     GO TO 1090
1080 LET cr=pfs: LET l$(cr)=r$:
     LET pfs=p(cr)
1090 LET ic=CODE r$(1)-64:
     LET stic=ic: IF pfr=0 THEN
     LET pfr=cr: LET a(ic)=cr:
     LET nr=0: GO TO 1190
1100 IF l$(pfr)>r$ THEN
     LET a(ic)=cr: LET nr=pfr:
     LET pfr=cr: GO TO 1190
1110 IF a(ic)<>0 THEN GO TO 1150
1120 LET a(ic)=cr
1130 LET ic=ic+1: IF ic<26 THEN
     IF a(ic)=0 THEN  GO TO 1130
1140 LET nr=a(ic): LET ic=stic:
    GO SUB 1300: GO TO 1180
1150 IF r$<l$(a(ic)) THEN
    LET nr=a(ic): LET a(ic)=cr:
    GO SUB 1300: GO TO 1180
1160 LET nr=a(ic)
1170 IF r$>=l$(nr) THEN
    LET pr=nr: LET nr=p(pr):
    IF nr<>0 THEN GO TO 1170
1180 LET p(pr)=cr
1190 LET p(cr)=nr: IF cambio THEN
    RETURN
1200 PRINT AT 20,4; FLASH 1;
    "Otra entrada (s/n)"
1300 REM Busqueda haci atras ****
1310 IF ic=1 AND cr=pfr THEN LET
     pr=0: RETURN
1315 IF ic=1 THEN LET ic=ic-1
      IF a(ic)=0 THEN GOTO 1315
1320 LET pr=a(ic):
     IF pr=0 THEN RETURN
1330 IF p(pr)<>0 THEN
     IF CODE l$(p(pr),1)-64=ic
     THEN LET pr=p(pr):
     GO TO 1330
1340 RETURN
```

Figura 6.11.

bético correspondiente al registro a insertar no es cero (ya hay un registro con la misma letra) la acción es menos complicada en lo que respecta al hallazgo del siguiente registro, pero una búsqueda hacia atrás sigue siendo necesaria si el nuevo registro desplaza al anterior a la cabeza de la subcadena. Un resultado interesante de todo esta actividad para unir enlaces es que en las etapas iniciales de la elaboración de un fichero, cuando pocos de los punteros alfabéticos no son cero, se experimentan un pequeño, pero no despreciable retardo. Cuando el fichero está más densamente "poblado", disminuirá el retardo, volviendo gradualmente más adelante cuando crezcan las subcadenas para cada letra del alfabeto.

6.6 Otros procedimientos (figuras 6.12 y 6.13)

La parte principal de la rutina de búsqueda es prácticamente invariable, pero lleva menos tiempo, pues comenzará por la mitad de la lista y se interrumpirá tan pronto como se encuentre un registro "más grande" que la cadena de búsqueda. Sin embargo, para poder encontrar los registros anterior y siguiente, que sean requeridos si el registro ha de suprimirse o modificarse, una búsqueda hacia atrás se llevará también a cabo. La lista de variables para el programa se da en la figura 6.14, que abarca también los dos programas anteriores.

6.7 Listas de variables

El programa utilizado para obtener listas de variables a través de este libro se da a continuación (figura 6.15), con las directrices correspondientes. Se ha esperado hasta este punto, porque muestra una lista encadenada en acción. Los valores de todas las variables utilizadas en un programa de BASIC se mantiene en memoria en un bloque, estando cada cadena o número inmediatamente precedido por la información acerca del tipo de variable y del número de octetos ocupados. Hay seis tipos de variables:

2. Cadena simple
3. Número cuya etiqueta, o rótulo, está constituido por una sola letra
4. Matriz de números
5. Número cuya etiqueta, o rótulo, está constituido por más de una letra
6. Matriz de cadena (una o más dimensiones)
7. Variable de control de bucle

```
4000 REM Encontrar una entrada ******
4010 PRINT AT 20,1; INVERSE 1;
     "Introduzca el nombre a encon
      trar a continuacion"
4020 INPUT "Nombre:";s$: PRINT
      AT 20,0;TAB 31;:
      LET ic=CODE s$(1)-64:
      GO SUB 800: IF NOT ok THEN
      GO TO 4010
4030 LET stic=ic: LET sl=LEN s$:
      LET cr=a(ic): GO SUB 1300:
      LET ic=stic: IF cr=0 THEN
      LET  encontrada=n; GO TO
      4060
4040 PRINT AT 200;TAB 31;
     AT 20,10; FLASH 1;
     "BUSQUEDA"
4050 IF cr<>0 THEN LET encon
     trada=s:
     IF s$<>l$(cr, TO sl) THEN
     LET encontrada=n: LET pr=cr:
     LET cr=p(cr):
     IF l$(cr,1)=s$(1) THEN
     GO TO 4050
4060 IF NOT encontrada THEN
     PRINT INVERSE 1; AT 8,3;
    "Ninguna coincidencia
    (mas) encontrada";:GO TO
    4190
4070 PRINT AT 8,11;"ENCONTRA
     DO"; AT 20,0;TAB 31
4080 PRINT AT 11,0; INVERSE 1;
    l$(cr);
4090 PRINT AT 20,4; FLASH 1;
    "Continuar busqueda (s/n)?"
4100 PAUSE 0: IF INKEY$="y" OR
     INKEY$="Y" THEN LET pr=cr:
     LET cr=p(cr): LET encon
     trada=n: GO TO 4050
4190 GO SUB 500: RETURN
5000 REM Imprimir la lista
5010 LET cr=pfr: LET i=3
5020 IF cr<>0 THEN
      PRINT AT i,0;l$(cr):
      LET cr=p(cr): LET i=i+1:
       IF i<18 THEN GO TO 5020
5040 PRINT AT i,0;TAB 31;" ":
     GO SUB 500
5050 LET i=3: IF cr<>0 THEN
      GO TO 5020
5060 RETURN
```

Figura 6.12.

```
2000 REM Suprimir una entrada ******
2010 GO SUB 4000:
      IF NOT encontrada THEN RE
      TURN
2020 PRINT AT 20,4; FLASH 1;
     " Pulse 'S' para borrar";
     AT 21,4;
     "cualquier otra tecla para
      retornar";
2030 PAUSE 0:
     PRINT AT 20,0;TAB 31;
    AT 21,0;TAB 31;
2040 IF INKEY$<>"y"
     AND INKEY$<>"Y" THEN RETURN
2050 LET rc=rc-1: LET nr=p(cr):
       LET pss=pfs: LET pfs=cr:
       LET p(cr)=pss
2060 IF pr=0 THEN LET pfr=nr
      GO TO 2080
2070 LET p(pr)=nr
2080 IF a(ic)<>cr
     THEN GO TO 2180
2090 IF nr<>0 THEN
      IF CODE l$(nr,1)-64<>ic
      THEN LET a(ic)=0:
       GO TO 2180
2100 LET a(ic)=nr
2130 GO SUB 500: RETURN
3000 REM Modificar una entrada
3010 GO SUB 4000: IF NOT encon
     trada THEN RETURN
3020 PRINT AT 20,2; INVERSE 1;
   "Introduzca cambio ahora"
    : INPUT "Nombre:"r$:
   PRINT AT 13,0; INVERSE 1;r$
   : GO SUB 600:
   IF NOT ok THEN GO TO 3020
3030 LET ic=CODE r$(1)-64:
     GO SUB 800: IF NOT ok THEN
     GO TO 3020
3040 PRINT AT 20,0;TAB 31;
     AT 20,4; FLASH 1;
    " Pulse 'S' para modificar";
    AT 21,4;
   "cualquier tecla para re
    tornar";
3050 PAUSE 0:
    PRINT AT 20,0;TAB 31;
    AT 21,0;TAB 31;.
    IF INKEY$<>"S" AND
     INKEY$<>"S" THEN RETURN
```

```
3060 GO SUB 2050: REM supresion
3070 LET cambio=s: GO SUB 1070:
     LET cambio=n: REM adicion
3080 PRINT AT 11,0; INVERSE 1;
     r$;AT 13,0;
     INVERSE 0;TAB 31;" "
3090 GO SUB 500: RETURN
```

Figura 6.13.

```
Lista de variables
------------------

cambio     Indicador-entrada a subrutina
y          Verdadero - 1
n          Falso - 0
i%         Contador de bucle
n$( )      Nombre (primer prog.)
t$( )      Num. tel (primer prog.)
ch         Eleccion de menu
ok         Indicador
l$( )      Matriz de registros
rc         Conteo de registros activos
r$( )      Registro de entrada
p( )       Punteros
a( )       Punteros alfabeticos
cont       Indicador (continuar)
pfs        Puntero primer reg. activo
pfr        Puntero primer reg. vivo
ec         Conteo de entradas
ic         Caracter inicial
cr         Registro actual
stic       Almac para siguiente ic
nr         Siguiente registro
pr         Registro anterior
sl         Longitud cadena busqueda
encontr    Indicador de busqueda
pss        Puntero segundo reg. muerto
s$         Cadena busqueda
m$         Descripcion de menu
a$         INKEY$
```

Figura 6.14.

Se han numerado de una forma peculiar de modo que el número anterior corresponda al valor de 5 en el programa, que, a su vez, se utiliza para indexar en una tabla de rutinas cortas (líneas 9836 a

```
9830 LET z1=PEEK 23641+256*PEEK
    23642-1: GO SUB 9855
9831 PRINT "Lista de variables":
     PRINT "______________________": IF z7
      THEN LPRINT "Lista de variables":
":      LPRINT "____________________"
9832 LET z2=0: LET z3=PEEK 23627
    +256*PEEK 23628
9833 DIM z$(10): LET z3=z3+z2: IF
    z3>z1 THEN STOP
9834 LET z4=PEEK z3: LET z5=4*(z
   4>127)+2*((z4>191) OR (z4>63 AND
   z4<128))+(z4>223 OR (z4>159 AND
   z4<192) OR (Z4>95 AND z4<128)):
   IF z5<2 OR z5>7 THEN PRINT "ER
      ROR": STOP
9835 LET z$=CHR$ (z4-32*z5+96):
   GO TO 9830+3*z5
9836 LET z$(2 TO )="$": LET z2=P
   EEK (z3+1)+256*PEEK (z3+2)+3: GO
   TO 9852
9839 LET z2=6: GO TO 9852
9842 LET z2=PEEK (z3+1)+256*PEEK
  (z3+2)+3: LET z$(2 TO )="( )":
  GO TO 9852
9845 LET z6=1
9846 LET z3=z3+1: LET z6=z6+1: L
    ET z4=PEEK (z3): LET z$(z6)=CHR$
     (z4-128*(z4>128)): IF z4<128 TH
  EN GO TO 9846
9847 LET z2=6: GO TO 9852
9848 LET z$(2 TO )="$( )": LET z
 2=PEEK (z3+1)+256*PEEK (z3+2)+3:
  GO TO 9852
9851 LET z$(2)="₧": LET z2=19
9852 PRINT : PRINT Z$;: IF z7 TH
 EN LPRINT z$;: DIM z$(22): INPU
 T "Comentario:-";z$;: IF z7 TH
  LPRINT z$
9853 GO TO 9833
9855 INPUT "Comentarios y salida
  impresa(s/n)";z$: let z7=(z$(1)
  ="s"):
  RETURN
```

Figura 6.15.

9848), que cálcula la cantidad de espacio ocupado por la variable que se examina. El programa actúa simplemente a través del espacio de memoria asignado a las variables, examinando cada una de ellas para ver cuanto espacio ocupa (para poder encontrar la siguiente) y examinando su etiqueta (rótulo) y tipo.

El programa propiamente dicho se presenta de una forma bastante diferente a todos los demás en este libro, debido a su función especial. Si se utiliza en la forma recomendada no generará informes sobre sus propias variables, sino solamente en el caso de que todos los rótulos en la rutina comiencen en la letra z. Está numerado con números altos de modo que pueda fundirse (MERGE) en otro programa sin perder ninguna de las líneas de dicho programa. Antes de utilizarlo, ha de dar a su programa un desarrollo para llevar a acción a todas las variables que utiliza; las variables no tienen asignado espacio de memoria a no ser que se hayan utilizado, y, a veces, hay líneas en un programa que en muy pocas ocasiones, si no nunca, son llamadas a acción. Si ha de realizar muchas modificaciones en su programa en el curso de su utilización, salvaguarde la nueva versión antes de su fusión en el programa de lista de variables. Teclee GOTO 983Ø (una ejecución —RUN— destruiría todas las variables). Con ello se da una oportunidad para conseguir una copia impresa, con anotaciones para referencia futura o también puede copiar (COPY) la pantalla. Si ha de utilizarla pora dos veces, todas las variables z serán objeto de listado.

6.8 Lista de existencias

En este programa final, se mostrará un esquema de organización muy similar al utilizado para las variables en la memoria de la computadora. El problema consiste en mantener una lista de artículos (elementos de datos) en memoria junto con las cantidades que se tiene en existencias, permitiendo así los procesos habituales de búsqueda, supresión y modificación. El último procedimiento implicará la suma o la resta del número mantenido en existencias. Solamente hay cuatro campos:

1. Descripción del artículo (elemento de datos)
2. Número de existencia (stock)
3. Nivel de reaprovisionamiento (cuando el número en existencias se hace inferior a este nivel, debe hacerse un nuevo pedido)
4. La fecha en que se hizo el pedido del artículo (ØØ/ØØ/ØØ si no está pendiente ningún pedido)

Ello representa el número mínimo de campos necesarios para un sistema de explotación, pero es la técnica empleada lo que más im-

porta. El número de caracteres utilizados en la descripción de un artículo puede variar mucho; para los artículos frecuentemente utilizados, sólamente bastaría la referencia más breve, pero para los demás podría ser preferible conservar una nota del proveedor y otros detalles como parte de la descripción. La asignación de un número determinado de caracteres para este campo constituiría un gran desperdicio de memoria de la computadora y reduciría el número de artículos que podríamos almacenar en un instante dado, porque todos los registros utilizarían la capacidad de almacenamiento utilizada por la descripción más larga.

Por consiguiente, la longitud de un registro se podrá variar, con el empleo de solamente el espacio suficiente para que se haga la entrada con unos pocos caracteres adicionales para suministrar información esencial. Ello significa que no podremos realizar la indexación en un fichero de la misma forma que se hizo tan frecuentemente a lo largo de este libro (todos los registros en una matriz de caracteres de Sinclair son de la misma longitud). Los registros se almacenarán ahora de extremo a extremo en una cadena unidimensional de longitud "fija" (una cadena de extensión simple aseguraría la clasificación de problemas que se trató en el capítulo 4). Cada registro llevará un quinto campo oculto situado antes de la descripción, que contendrá la longitud de registro convertida en una cadena de dos caracteres. Por este motivo, se supondrá que la longitud de registro máxima es de 99 caracteres, lo que constituye una restricción que se podría suprimir con facilidad. Una vez que hayamos encontrado un registro, el siguiente se localiza mediante un salto sobre ese número de caracteres. En la figura 6.16 se muestra con detalle cómo se almacena un registro.

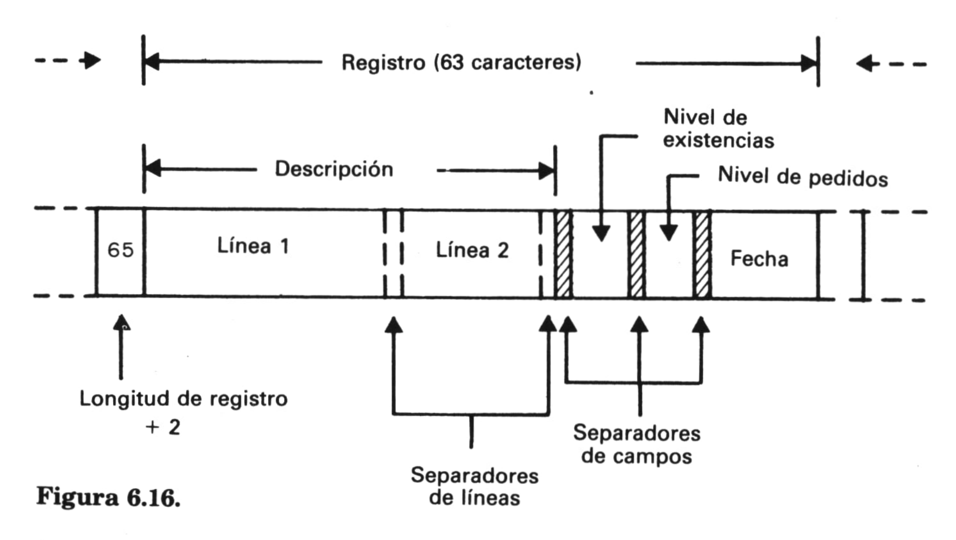


Figura 6.16.

Los tres primeros campos (descripción, nivel de existencias y nivel de pedidos) son de longitud variable y ello se consigue con el empleo de un separador de campos. Este último es un carácter único que actúa solamente como una clasificación de indicador. Puesto que no debe aparecer en cualquier lugar en el fichero, se eligió uno de los códigos de control, CHR$ 6, para esta función. El campo de la fecha tendrá siempre 8 caracteres de longitud sin que se requiera ningún separador. El campo de descripción estará constituido por dos líneas de longitud variable y un separador de línea indica el final de cada una. Como una línea se suele finalizar con CHR$ 13, la tecla ENTER, esta última será la utilizada. Tiene el efecto secundario útil de que cuando la descripción es objeto de impresión como una sola unidad, el CHR$ 13 actuará como un código de control y dividirá la descripción en dos líneas en la pantalla, tal como se introdujo originalmente. La utilización de los separadores significa que no podemos almacenar nada más que el número de caracteres introducidos, pero este esquema organizativo seguirá siendo más económico que los acampo de longitud fija. Los separadores se suelen denominar también delimitadores.

Para poder atender al requisito de los campos de longitud variable, se elaboró una nueva rutina de entrada en pantalla, que se ilustra en la figura 6.17. Permite que se utilice solamente teclas de caracteres normales junto con las de DELETE Y ENTER, efec-

```
 10 REM .............................................
..........................
100 REM Linea de entrada *****
110 DIM i$(32): LET y=0:
    LET il=1
120 LET y=y*(y>=0)-(y>31):
    PRINT AT x,0; PAPER 6;i$;
    AT x,y; PAPER 4;i$(y+1)
130 PAUSE 0: LET c=PEEK 23560:
    LET a$=CHR$ c: BEEP .004,0:
    IF c>31 AND c<128 THEN
    LET i$(y+1)=a$: LET y=y+1:
    LET il=il+(y>il): GO TO 120
140 IF c=12 THEN
    LET i$(y+(y=0))=" ":
   LET i$=i$( TO y)+i$(y+2 TO ):
   LET y=y-1: LET il=il-(il>1):
   GO TO 120
150 IF c<>13 THEN GO TO 120
160 PRINT AT x,y; PAPER 6;"<":
    RETURN
```

Figura 6.17

túa el conteo del número de caracteres tecleados (permitiendo los borrados) e imprime un "< " al final de la entrada. Actúa con mayor rapidez que las dos presentadas en el capítulo 4, pero utiliza un método muy similar. Para su empleo, el número de la línea x debe pasarse a la subrutina y proporciona una cadena i$ con el número de caracteres en una variable il, por lo que la cadena que ha de almacenarse se da por i$(TO il).

En la rutina a añadir un registro, mostrada en la figura 6.18, la rutina de entrada de línea es objeto de llamada repetida hasta que, en las líneas 1230 y 1240, el registro pueda juntarse con sus separadores. La cadena que mantiene su longitud se añade al principio de la línea 1250, y finalmente, la cadena completa se fragmenta en

```
1000 REM Añadir registro *********
1010 PRINT AT 5,0;
   m$(1);" etc (3 lineas)":
   LET d$=""
1040 FOR i=1 TO 2: LET x=5+i:
     GO SUB 100:
     LET d$=d$+i$( TO il)+e$:
     NEXT i
1060 PRINT AT 8,0;m$(2):
  LET x=9: GO SUB 100:
  LET n$=i$( TO il)
1070 PRINT AT 10,0;m$(3):
     LET x=11: GO SUB 100:
     LET o$=i$( TO il)
1080 PRINT AT 12,0;m$(4):
     LET x=13: GO SUB 100
1090 LET f$=i$( TO 8):
     IF f$="        " THEN
     LET f$="00/00/00"
1200 PRINT AT 20,2; FLASH 1;
 " Conforme entrada (s/n) "
1210 PAUSE 0: IF INKEY$<>"s" AND
      INKEY$<>"n" THEN BEEP .2,40
     : GO TO 1210
1220 IF INKEY$="n" THEN
      GO TO 1010
1230 LET r$=d$c$n$c$o$c$+
     f$( TO 8)
1240 LET rl=LEN r$+2:
     LET r$=STR$ rl+r$
1250 LET l$(ll TO ll+rl-1)=r$:
     LET ll=ll+rl
1260 RETURN
```

Figura 6.18.

la cadena unidimensional de registros. Si no se introduce ninguna fecha, la cadena "ØØ/ØØ/ØØ" se almacena en ese campo. De cualquier otro modo, no habrá ninguna comprobación y sería recomendable utilizar la pequeña rutina dada en el capítulo 4 para la validación de la entrada. Los niveles de existencias y de pedidos se podrían comprobar también utilizando uno u otro de los programas de validación en números incluidos en ese capítulo (la rutina de código de máquina utiliza menos espacio)

En la figura 6.19 se muestra la subrutina de menú y de utilidad única. Desde el punto de vista del usuario, una respuesta en letras

```
220 CLS>:BORDER 3:
     PRINT AT 2,9; INVERSE 1;
  "  Lista de existencias"
210 PRINT AT 5,5;
    "a - añadir un registro"
220 PRINT AT 7,5;
    "c - cambios niveles existen
   cias"
230 PRINT AT 9,5;
    "f - hallar un registro"
240 PRINT AT 11,5;
    "p - imprimir la lista"
250 PRINT AT 13,5;
    "d - suprimir un registro"
260 PRINT AT 15,5;
    "o - articulos a pedir"
270 PRINT AT 20,4; FLASH 1;
    "Pulse tecla de su elec
    cion"
280 PAUSE Ø: LET a$=INKEY$:
     LET l=300+700*(a$="a")+
    1700*(a$="c")+2700*(a$="f")
   +3700*(a$="p")+4700*(a$="d")
   +5700*(a$="o")
290 CLS : BORDER 3:
    PRINT AT 2,9; INVERSE 1;
    "Lista existencias":
    GO SUB 1: GO TO 200
300 BEEP .2,40: BEEP .2,20:
    RETURN
900>REM Patkc ******************
910 PRINT AT 20,3; FLASH 1;
   "Pulse cualquier tecla para
   continuar"
920 PAUSE Ø:
    PRINT AT 20,Ø;TAB 31:
    RETURN
```

Figura 6.19.

es más fácil de recordar que los números utilizados hasta ahora, pero plantea problemas de indexación, por lo que finalizamos con una línea bastante larga para esta finalidad en la línea **200**. Obsérvese que si no se da ninguna de las respuestas sugeridas, el valor de 1 será **300**, con lo que se dará la señal acústica de aviso habitual (BEEP) antes de volver al menú.

El lector puede haber adivinado a partir de la sentencia en REM, en la línea 10 en el listado de la figura 6.17, que una corta rutina en código de máquina está implicada en este programa. La preparación de esta rutina y de la cadena que contiene los registros se muestra en la figura 6.20 (no en el menú). La cadena de longitud fija se inicializa en la línea **7100**, que se puede utilizar sin las líneas precedentes, todas las cuales se pueden suprimir después de haberse utilizado primero, si el espacio es realmente un objetivo a conseguir.

La dimensión correspondiente a l$ podría ser mucho más grande en una máquina de 48K. La rutina en código de máquina se utiliza en el borrado de registros. Para dejar libre el espacio utilizado por un registro en la parte media de la matriz, los que siguen se desplazan hacia abajo en un bloque para su cubrimiento en sentido ascendente. En BASIC, ello traería consigo una línea de programa semejante a:

5040 LET l$ = l$(TO rs) + l$(re + 1 TO)

```
7000 REM Comienzo *************
7010 DIM m$(4,11)
7020 LET m$(1)="Descripcion":
     LET m$(2)="Nivel existen
     -cias"
7030 LET m$(3)="Nivel pedidos":
     LET m$(4)="Fecha pedido"
7070 LET l1=1: LET c$=CHR$ 6:
   LET e$=CHR$ 13: RESTORE 7090
7080 FOR i=23760 TO 23780:
     READ b: POKE i,b: NEXT i
7090 DATA 0,0,0,0,0,0,
          237,75,210,92,237
7091 DATA 91,208,92,42,212,
          92,25,237,176,201
7100 REM Arranque en caliente ***
7110 LET maxl=3000: DIN l$(maxl)
     : GO TO 200
8000 REM Conservar prog/datos ***
8010 SAVE "Lista existencias"
     LINE 200: STOP
```

Figura 6.20.

en donde rs y re marcan el principio y el final del registro no deseado. Ello implicaría la construcción de cadenas grandes en el espacio de trabajo, con lo que se reduciría la cantidad máxima de información que se podría mantener en aproximadamente la mitad del máximo teórico. El programa en código máquina realiza el mismo proceso que la línea BASIC, pero no utiliza ningún espacio de trabajo, porque funciona completamente dentro del espacio asignado a la variable 1$. En la línea 5050 se encuentra la posición del registro a suprimir y se almacena para uso posterior por el programa en código de máquina; a continuación, en la línea 5070 y 5080, se trata de forma similar la longitud de registro y el número de octetos a desplazar. En una circunstancia muy rara, el empleo de la rutina en código máquina impide que se suprima un registro. Si obtiene el mensaje mostrado en la línea 5060 (figura 6.21), tendrá que utilizar el método mostrado en el Apéndice para ampliar la longitud de la cadena, si fuera posible, o si está utilizando la memoria completa, aceptar que no se pueda almacenar nada más. No

```
5000 REM Supresion **************
5010 GO SUB 3000: IF rm>11 THEN
     RETURN
5020 PRINT AT 20,4; FLASH 1;
     "Pulse "s" para suprimir";
    AT 21,4;
    "cualquier otra tecla para re
    tornar";
5030 PAUSE 0:
     PRINT AT 20,0;TAB 31;
    AT 21,0;TAB 31;
5040 IF INKEY$<>"S"
     AND INKEY$<>"s" THEN RETURN
5050 LET l$(rm-2)=l$(rm-2):
     POKE 23760,PEEK 23629:
     POKE 23761,PEEK 23630
5060 LET nb=maxl-(rm+rl-2):
      IF nb<=0 THEN
      PRINT AT 19,4; FLASH 1;
     "El registro no se puede
      borrar":
     GO SUB 900: RETURN
5070 POKE 23763,INT (nb/256):
  POKE 23762,nb-256*INT (nb/256):
5080 POKE 23765,INT (rl/256):
   POKE 23764,rl-256*INT (rl/256)
5090 RANDOMIZE USR 23766:
      LET ll=ll-rl: RETURN
```

Figura 6.21.

impedirá más operaciones futuras, puesto que solamente se presenta si trata de borrar un registro muy cerca del final de la cadena dimensionada.

Como es habitual, las rutinas utilizadas para encontrar y visualizar un registro son esenciales en otros procesos. El listado en la figura 6.22 realiza una búsqueda en la manera utilizada en otros programas (sobre el número de caracteres introducidos por el usuario). Las variables rm y rl se utilizan para indexar en la matriz de cadenas, y se emplean tres procedimientos cortos, pero consumidores de tiempo, mostrados en la figura 6.23, para dividir el registro en sus campos y visualizarles. Un perfeccionamiento, en lo que respecta a la velocidad, se podría realizar si el registro se almacenó con sus campos más cortos al principio, y quizá al lec-

```
3000 REM Encontrar registro ****
3010 CLS : PRINT AT 15,0; "
Sirvase introducir la descrip
cion del producto a continua
cion. Teclee lo suficiente pa
ra obtener una identificacion
singular - el nombre completo
puede no ser necesario."
3020 INPUT "Descripcion":s$:
    CLS : PRINT AT 18,0;
     "Descripcion:"s$:TAB 31;
3030 PRINT AT 20,11; FLASH 1;
     "Busqueda":
     LET rm=3: LET sl=LEN s$
3040 LET rl=VAL l$(rm-2 TO rm-1)
     : IF l$(rm TO rm+sl-1)<>s$
     THEN LET rm=rm+rl:
      GO TO 3040+100*(rm>11)
3050 PRINT AT 2,13; INVERSE 1;
     "Encontrado ": BEEP .2,0
3060 GO SUB 500: GO SUB 600
3080 PRINT AT 20,4; FLASH 1;
     "Continuar busqueda
      (s/n)?"
3100 PAUSE 0: IF INKEY$="s" OR
     INKEY$="S" THEN
      LET rm=rm+rl:
       GO TO 3040+100*(rm>11)
3120 GO SUB 900: RETURN
3140 PRINT AT 2,11; INVERSE 1;
     " No hallado": GO SUB 900
     : RETURN
```

Figura 6.22.

```
400>REM Desplazar a lo largo ****
410 LET fr=fr+1:
    IF l$(fr)<>CHR$ 6 THEN
    GO TO 410
420 LET t$=l$(fl TO fr-1):
    LET fl=fr+1:LET fr=fl:
    RETURN
500 REM Dividir registro *********
510 LET fl=rm: LET fr=rm
520 GO SUB 400: LET d$=t$
530 GO SUB 400: LET n$=t$
540 GO SUB 400: LET o$=t$:
    LET f$=l$(fl TO fl+7)
550 RETURN
600 REM Visualizacion registro ***
610 FOR i=4 TO 13: PRINT AT i=0;
    TAB 31;" ":NEXT i
620 PRINT AT 5,0;"Descr:"
    PAPER 6;d$; PAPER 7;TAB 31
630 PRINT AT 8,0;"Exist:"
    PAPER 6,n$; PAPER 7;TAB 31
640 PRINT AT 10,0;"Nivel:"
    PAPER 6;o$; PAPER 7;TAB 31;
650 PRINT AT 12,0;"Pedido:"
   PAPER 6;f$; PAPER 7: RETURN
```

Figura 6.23.

tor le pueda agradar intentarlo. La rutina en la línea 400 busca a lo largo del registro para encontrar un separador de campos y si el campo de descripciones es largo, ello proporciona un retardo digno de consideración. Una alternativa sería dejar a los campos en su orden actual y buscar hacia atrás desde el final.

El retardo es más notable cuando ha de decodificarse cada registro, que es el caso que se muestra en las rutinas de la figura 6.24. La comprobación de si un artículo ha de ser objeto de pedido implica el examen de los niveles de existencias y de reaprovisionamiento. Si las existencias son demasiado bajas y "00/00/00" indica que no se ha hecho ningún pedido, se visualizará el registro; de no ser así, podemos pasar al siguiente. En la segunda rutina, se visualiza cada uno de los registros. Ambas se podrían modificar para enviar salida a una impresora.

En la figura 6.25. se muestra una rutina que permite cambiar el nivel de existencias y la fecha del último pedido. Se introducen los cambios con el empleo de una sentencia INPUT y se visualiza en la pantalla junto con los valores antiguos y luego, la interrogante habitual efectúa la petición de que se confirme que se deben realizar los cambios correspondientes. En el programa, el cambio al ni-

```
4000 REM Impresion ***********
4010 LET rm=3
4020 PRINT AT 20,11; FLASH 1;
     "Busqueda":
    LET rl=VAL l$(rm-2 TO rm-1)
    : GO SUB 500: GO SUB 600
4030 LET rm=rm+rl: GO SUB 900:
     IF rm<11 THEN GO TO 4020
4040 RETURN
6000 REM Articulos a pedir *****
6010 LET rm=3: LET oc=o
6020 PRINT AT 20,11; FLASH 1;
     "Busqueda"
     LET rl=VAL l$ (rm-2 TO rm-1)
    :GO SUB 500
6030 IF VAL n$<=VAL o$ AND
     f$="00/00/00" THEN
     GO SUB 600: LET oc=oc+1:
     GO SUB 900
6040 LET rm=rm+rl:
     IF rm<11 THEN GO TO 6020
6050 PRINT AT 19,6; FLASH 1;
oc;"  articulo(s) a pedir "
6060 GO SUB 900:
     PRINT AT 19,0;TAB 31;
6100 RETURN
```

Figura 6.24.

vel de existencias se efectúa convirtiendo los valores de cadenas en valores numéricos con la utilización de VAL. En este caso, es fundamental cambiar el registro real suprimiendo la versión antigua y añadiendo una versión nueva puesto que la longitud real se podría haber cambiado y el registro modificado haber llegado a ser demasiado grande para encajar en el espacio anteriormente utilizado.

6.9 ¿A dónde ahora?

En el último programa, se hicieron varias sugerencias para perfeccionamientos. Una mejora adicional podría hacerse utilizando una forma de localización de lista encadenada, sobre todo si se utilizaron punteros alfabéticos, pues solamente cuando dichos punteros se incluyen se reduce, de forma notable, los tiempos de búsqueda. Sin embargo, para los punteros simples, sería muy importante almacenarles con los registros en otro campo "oculto" similar al utilizado para las longitudes de registros. En realidad, el hecho de seguir almacenando las longitudes de los registros sería un desperdicio de espacio si el puntero diera un índice directo para el siguiente registro

```
2000 REM Niveles de existencias ***
2010 GO SUB 3000: IF rm>11 THEN
      RETURN
2020 PRINT AT 20,0; INVERSE 1; "
     Dar cambio (+ 0 -) en stock"
2030 INPUT "Cambio:";a$:
     PRINT AT 20,0; TAB 31;
     AT 8,10; FLASH 1;a$
2040 INPUT (m$(4));f$: IF f$=""
    THEN LET f$="00/00/00"
2050 PRINT AT 12,10; FLASH 1;f$
2060 PRINT AT 20,2; FLASH 1;
     "Conforme entrada (s/n)"
2070 PAUSE 0:
     PRINT AT 20,0;TAB 31;:
   IF INKEY$<>"s" AND INKEY$<>"n"
    THEN BEEP .2,40: GO TO 2060
2080 IF INKEY$="n" THEN
      GOTO 2020
2090 PRINT AT 20,4; FLASH 1;
     "Pulse ´ ´para cambiar ";
     AT 21,4;
     "cualquier otra tecla para
     retornar";
2100 PAUSE 0:
      PRINT AT 20,0; TAB 31;
     AT 21,0; TAB 31;
2110 IF INKEY$<>"S"
      AND INKEY$<>"s" THEN RETURN
2120 GO SUB 5050
2130 LET n$=STR$ (VAL n$+VAL a$)
      : PRINT AT 20,0;TAB 31:
      GO SUB 1230
2140 RETURN
```

Figura 6.25.

en la secuencia “lógica”. Sin embargo, hay que tener presente que el espacio adicional ocupado por el propio programa reduciría, de forma considerable, el número de registros almacenados.

Espero que algunos de, o todos, los programas dados en los últimos capítulos habrán satisfecho sus propias necesidades. De no ser así, debe ser posible utilizar las diversas partes componentes para elaborar algo que se adapte a las necesidades individuales. Cuando este libro salga de imprenta lanza al mercado la unidad Microdrive y si usted la pretende utilizar para aumentar el tamaño del fichero que puede almacenar, serán de utilidad muchas de las técnicas utilizadas en este libro. En cualquier caso, espero poder abordar este tema en un trabajo posterior (¡siempre que logre la tolerancia del editor!).

APENDICE

Aumento de la magnitud de las matrices de “tamaño fijo”

1. Ha de cerciorarse de que tiene una copia de reserva de su programa con sus variables.
2. Ha de calcular cuidadosamente el tamaño de la matriz. Una matriz de caracteres utiliza un octeto por elemento, por lo que una operación, por ejemplo, DIM 1$(24,36) utiliza 24*36 = 864 octetos. Una matriz de números utiliza cinco octetos por entrada, por lo que DIM n(24,36) utilizaría hasta 24*36*5 = 4320 octetos.
3. Introduzca una de las siguientes líneas en su programa, dependiendo de si su matriz es de caracteres o numérica. Ha de tener presente que debe utilizar su propio rótulo (etiqueta) de variables y el número correcto de dimensiones.

   ```
   9888 LET l$(1):PRINT PEEK 23629 + 256*PEEK 23630:
            STOP
   9888 LET n(1) = n(1):PRINT PEEK 23629 + 256*PEEK 23630:
            STOP
   ```
4. Teclee GOTO 9888 (o cualquier otro número de línea que utilice) y haga una anotación cuidadosa del número que se imprime en la pantalla.
5. No haga ahora una edición del programa ni lo ejecute, pues ello cambiaría la posición de la matriz (el número en esa posición).
6. Introduzca por el teclado (sin ningún número de línea):

   ```
   SAVE“bloquemem”CODE 99999,999
   ```
   en donde los dos números son, respectivamente, la posición de la matriz y su tamaño según se calculó anteriormente. Se puede verificar la función SAVE: bastará VERIFY” CODE. Ahora redimensione la matriz tecleando DIM1$(64,36). Con ello se borra todos los datos antiguos de la memoria, pero sigue estando en la cinta. Por supuesto, debe hacer la nueva matriz más grande que la antigua y a menos que comprenda exactamente lo que está haciendo, la última dimensión debe ser exáctamente la misma que la anterior.
7. Repita el paso 4 y de nuevo haga una anotación cuidadosa del número.

8. Teclee (de nuevo sin número de línea)

   LOAD“bloquemem”CODE 88888

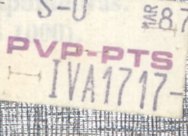
   en donde el número es encontrado en el paso 7. No hay ninguna necesidad de introducir un segundo valor, pero si desea hacerlo debe ser el mismo que el segundo en el paso 6. La matriz debe contener ahora todos los datos antiguos y tener un espacio suplementario para la expansión.

Manipulación de la información del Spectrum

Una de las aplicaciones más importantes de las microcomputadoras está en la manipulación de la información. Con más de 40K de RAM del usuario, el Spectrum es más que suficiente para atender a la mayor parte de las actividades domésticas o de los pequeños negocios. Y la programación para la manipulación de datos no es difícil, una vez que se hayan conocido los principios fundamentales.

Técnicas estructuradas

En este libro, las técnicas de programación estructuradas se utilizan para desarrollar programas de manipulación de la información de la vida real. Treinta programas acabados y rutinas son objeto de listado; muchos de ellos son útiles en sí mismos y todos ellos son adecuados para su inclusión en sus propios programas adaptados a sus necesidades individuales.



ISBN: 968-451-725-4

